

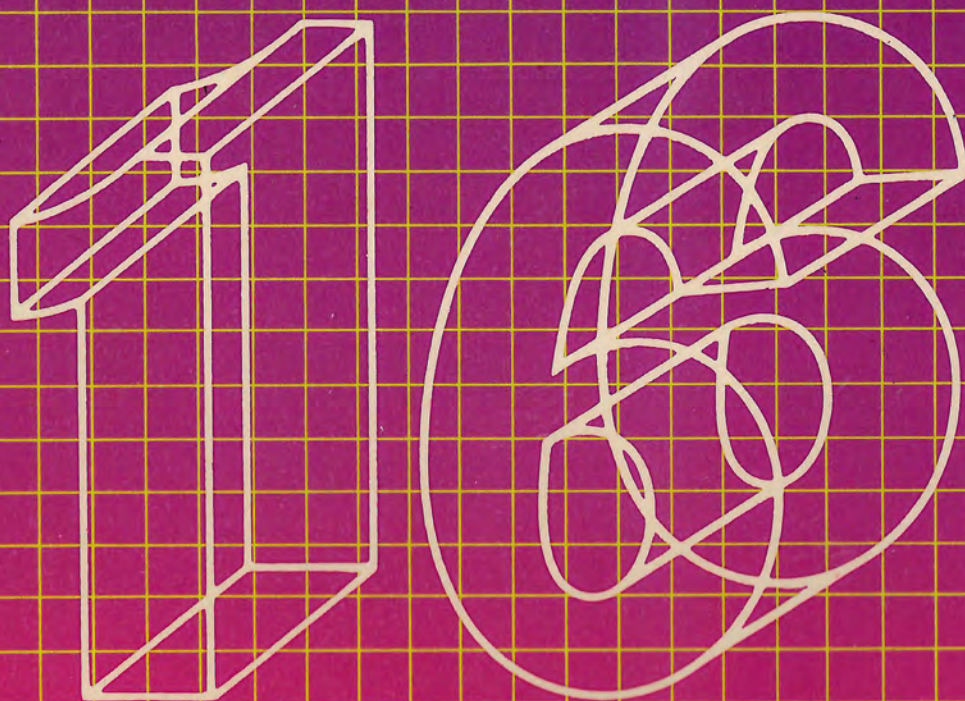
BIBLIOTECA PRACTICA

TRUCOS - SPRITES
BRICOLAGE DEL HARD
AULA ABIERTA
LOGO - TERMINOLOGIA

TALLER DE INFORMATICA

PROGRAMAS VALIDOS PARA AMSTRAD,

IBM, SPECTRUM, COMMODORE Y MSX



PROGRAMAS PRACTICOS

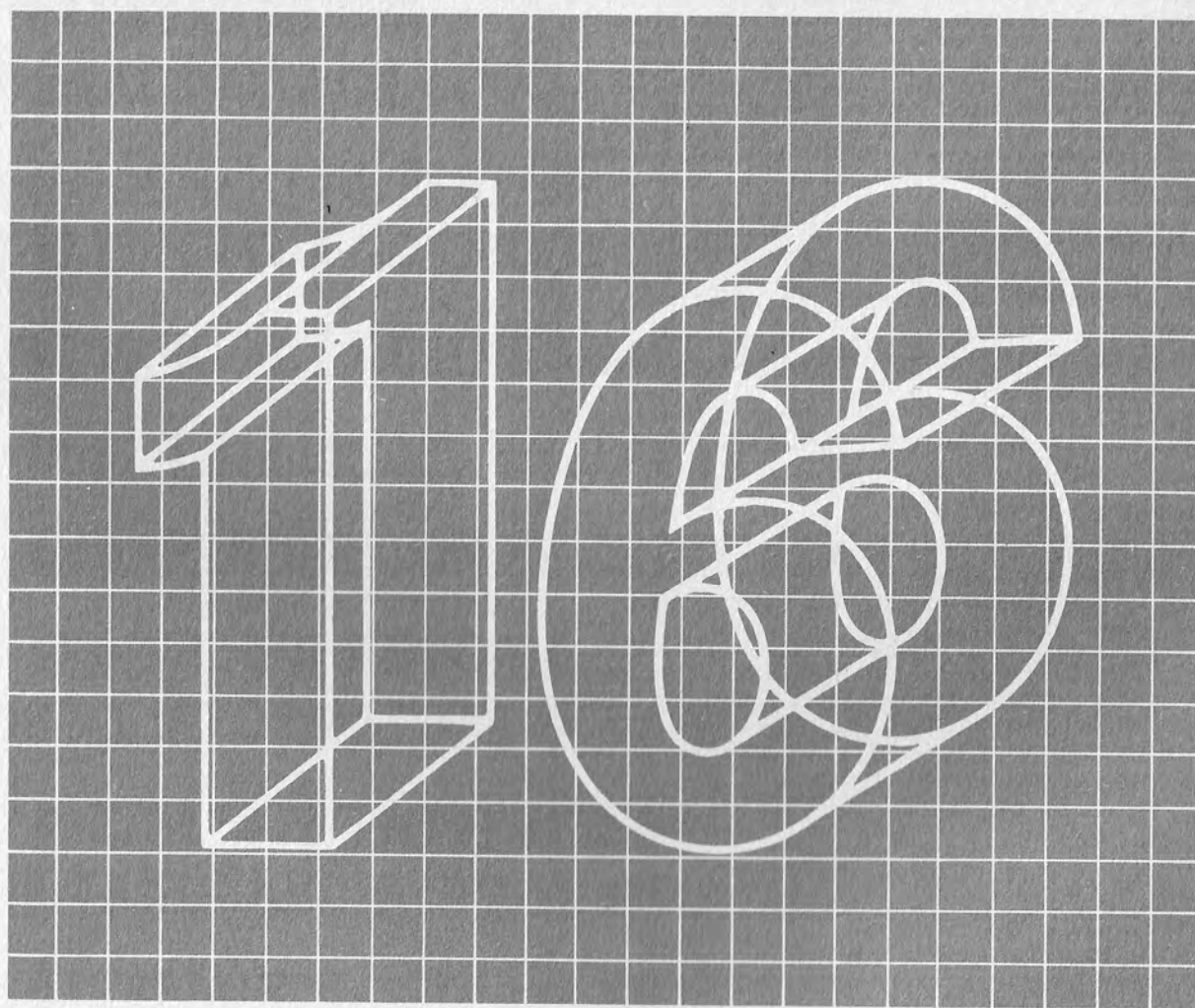
TRUCOS, BRICOLAGE Y GRAFICOS

EDICIONES SIGLO CULTURAL

BIBLIOTECA PRACTICA TALLER DE INFORMATICA

TRUCOS - SPRITES
BRICOLAGE DEL HARD
AULA ABIERTA
LOGO - TERMINOLOGIA

PROGRAMAS VALIDOS PARA AMSTRAD,
IBM, SPECTRUM, COMMODORE Y MSX



EDICIONES SIGLO CULTURAL

Una publicación de

EDICIONES SIGLO CULTURAL, S. A.

Director-editor:

RICARDO ESPAÑOL CRESPO.

Gerente:

ANTONIO G. CUERPO.

Directora de producción:

MARIA LUISA SUAREZ PEREZ.

Director de la colección:

JOSE ARTECHE, Ingeniero de Telecomunicación

Diseño:

BRAVO-LOFISH.

Maquetación:

D. S. M.

Dibujos:

JOSE OCHOA

Tomo 16. «Experiencias y prácticas en logo», Carlos Albert, Técnico de Informática; Adoración Llena Jubero, Técnico de Informática. «Manejo de sprites y elementos gráficos», «Trucos y rutinas básicas», Francisco Morales. Técnico de Informática. «Aprende con el ordenador», AULA DE INFORMATICA APLICADA: Soledad Tamariz-Martel, Diplomada en Telecomunicación; Francisco Blanca, Diplomado en Telecomunicación; Alejandro Marcos, Licenciado en Química; Fernando Suero, Diplomado en Telecomunicación. «El Taller del Hardware», Carlos Rey, Ingeniero Industrial. «Pequeña historia de la Informática», «Temas monográficos de vanguardia», «Terminología», «Vocabulario de Informática», Blanca Arbizu, Técnico de Informática.

Ediciones Siglo Cultural, S. A.

Dirección, redacción y administración:

Pedro Teixeira, 8-2.º. Teléf. 810 52 13. 28020 Madrid.

Publicidad:

Gofar Publicidad, S. A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:

COEDIS, S. A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.

Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:

CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.

Buenos Aires - 1.290. Argentina.

Todos los derechos reservados. Este libro no puede ser, en parte o totalmente, reproducido, memorizado en sistemas de archivo, o transmitido en cualquier forma o medio, electrónico, mecánico, fotocopia o cualquier otro, sin la previa autorización del editor.

ISBN del tomo: 84-7688-087-1

ISBN de la obra: 84-7688-047-2

Fotocomposición:

ARTECOMP, S. A. Albarracín, 50. 28037 Madrid.

Imprime:

MATEU CROMO. PINTO (Madrid).

© Ediciones Siglo Cultural, S. A., 1986

Depósito legal: M-43.185-1986

Printed in Spain - Impreso en España.

Suscripciones y números atrasados:

Ediciones Siglo Cultural, S. A.

Pedro Teixeira, 8-2.º. Teléf. 810 52 13. 28020 Madrid.

Mayo, 1987

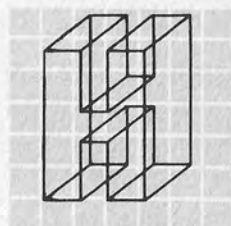
P.V.P. Canarias: 355.—Ptas.

INDICE

■ EXPERIENCIA Y PRACTICAS EN LOGO	5
■ MANEJO DE SPRITES Y ELEMENTOS GRAFICOS	14
■ TRUCOS Y RUTINAS BASICAS	21
■ EL TALLER DEL HARDWARE	28
■ APRENDER CON EL ORDENADOR	35
■ PEQUEÑA HISTORIA DE LA INFORMATICA	45
■ TEMAS MONOGRAFICOS DE VANGUARDIA	49
■ TERMINOLOGIA	55
■ VOCABULARIO DE INFORMATICA	57

EXPERIENCIA Y PRACTICAS EN LOGO

■ Expresiones lógicas



EMOS visto que una expresión relacional devuelve siempre el resultado CIERTO o el resultado FALSO; si las operaciones que devuelven alguno de estos dos resultados lógicos reciben el nombre de predicados concluimos que una expresión relacional es un predicado.

Los operandos de una expresión lógica son predicados.

Y los operadores son:

NO predicado1 predicado2

Y predicado1 predicado2

O predicado1 predicado2

El resultado de una expresión lógica es la palabra CIERTO o la palabra FALSO; por tanto, estas expresiones son también predicados.

— OPERACION NO

NO predicado1

Devuelve CIERTO si predicado es FALSO y devuelve FALSO si predicado es CIERTO; es decir, devuelve el valor lógico contrario al del predicado.

(predicado1) es el dato de la expresión NO.

La operación NO sólo puede tener un dato.

A continuación mostramos la tabla de la operación NO.

1 = CIERTO.

0 = FALSO.

D	-	R
1	-	0
0	-	1

Ejemplos:

? ESCRIBE NO "CIERTO
FALSO

? ESCRIBE NO "FALSO
CIERTO

? ESCRIBE NO (2 > 3)
CIERTO

? ESCRIBE NO (2000 / 10) < 720
CIERTO

? ESCRIBE NO (MENOR? 5 (10 / 2))
CIERTO

? HAZ "A "CIERTO

? ESCRIBE NO :A
FALSO

— OPERACION Y

Y predicado1 predicado2.

Devuelve CIERTO si y sólo si todos sus datos son ciertos, si no devuelve FALSO.

(predicado1 y predicado2) son los datos de la expresión.

El número de datos de la operación Y puede ser variable.

A continuación mostramos la tabla de la operación Y con dos datos.

1 = CIERTO

0 = FALSO

Los operadores de relación sirven para comparar dos objetos.

D	D	-	R
1	1	-	1
1	0	-	0
0	1	-	0
0	0	-	0

Ejemplos:

```
? ESCRIBE Y "CIERTO
CIERTO
? ESCRIBE Y (2 < 10) "CIERTO
CIERTO
? ESCRIBE Y (2 > 3) (4 < 2)
FALSO
? ESCRIBE Y ((2000 / 33) < 720) ((20 *
43) < 720)
FALSO
? ESCRIBE O (MENOR? 5 (10 / 2))
(IGUALES? 5 (10 / 2))
FALSO
? HAZ "A CIERTO
? HAZ "B FALSO
? ESCRIBE Y :A :B
FALSO
? HAZ "A (30*80) < 200
? HAZ "B (200 / 3) > 65
? ESCRIBE Y :A :B
FALSO
— OPERACION O
O predicado1 predicado2
```

Devuelve FALSO si sus datos son todos FALSO, si no devuelve CIERTO.

El número de datos de esta operación puede ser variable.

La tabla de esta operación con dos datos es:

D	D	-	R
1	1	-	1
1	0	-	1
0	1	-	1
0	0	-	0

Ejemplos:

```
? ESCRIBE O "CIERTO "FALSO
CIERTO
? ESCRIBE O "FALSO "FALSO
FALSO
? ESCRIBE O (MENOR? 5 (10 / 2))
(IGUALES? 5 (10 / 2))
CIERTO
? HAZ "A "CIERTO
```

```
? HAZ "B "CIERTO
? ESCRIBE O :A :B
CIERTO
? HAZ "A 300
? HAZ "B 30
? HAZ "C :A > :B
? HAZ "D (:A / 10) = :B
? ESCRIBE :C
CIERTO
? ESCRIBE :D
CIERTO
? ESCRIBE O :C :D
CIERTO
```

Operadores de relación

Los operadores de relación se utilizan para comparar dos objetos, el resultado de la comparación sólo puede ser CIERTO o FALSO.

En LOGO disponemos de tres de estos operadores:

```
MAYOR?
MENOR?
IGUAL?
— MAYOR? n1 n2
```

Preguntamos 'al LOGO si n1 es mayor que n2. Puede escribirse también utilizando el signo >, que significa mayor.

```
> n1 n2 forma prefija
n1 > n2 forma infija
? ESCRIBE MAYOR? 5 7
FALSO
? ESCRIBE MAYOR? 10 2
CIERTO
? ESCRIBE > -1 -2
CIERTO
? ESCRIBE 0 > 3
FALSO
? ESCRIBE (20+4) > (.4 * 10)
CIERTO
? HAZ "A 30
? HAZ "B 50
? HAZ "C 4
? ESCRIBE ((:A * :B) - :B * :B) > (:A + :B)
FALSO
```

Recuerda que el resultado de una expresión aritmética es siempre un objeto.

Las operaciones que devuelven el valor lógico cierto o falso reciben el nombre de predicados.

— MENOR? n1 n2
¿Es n1 menor que n2?

También podemos escribirlo así:

< n1 n2 forma prefija
n1 < n2 forma infija
? ESCRIBE MENOR? 7 20
CIERTO
? ESCRIBE < 20 3
FALSO
? ESCRIBE < 40 100
CIERTO
? ESCRIBE 19 < 0
FALSO
? HAZ "C 50
? ESCRIBE 20 < :C
CIERTO
? HAZ "A 1299
? HAZ "B 2929
? ESCRIBE MENOR? (:A / :B) (:A - (:B /
10))
CIERTO

— IGUALES? n1 n2
¿Son iguales n1 y n2?
Esta expresión es equivalente a:

= n1 n2 forma prefija
n1 = n2 forma infija
? ESCRIBE IGUALES? 20 20
CIERTO
? ESCRIBE = 30 50
FALSO
? ESCRIBE 50 = 50
CIERTO
? HAZ "A 1010
? HAZ "B 2020
? ESCRIBE (:A * 2) = :B
CIERTO

es lo mismo que escribir

? ESCRIBE IGUALES? (1010 * 2) 2020

■ Gestión con variables

En ocasiones es necesario saber cuáles son las variables que se están utilizando en un proceso. Esto puede ser de gran ayuda si en

un momento determinado queremos utilizar una nueva variable y no recordamos si ya está utilizada.

Para esto el Logo dispone de una orden que nos lo dice:

VAR? "Nombre de variable

Si asignamos un valor a la variable I:

? HAZ "I 20

y a continuación introducimos:

? ES VAR? "I

en la pantalla aparecerá el mensaje:

CIERTO

Es decir, nos dice que I sí es una variable.

Si ahora introducimos:

? ES VAR? "II

en la pantalla aparecerá el mensaje:

FALSO

siempre y cuando no hayamos asignado anteriormente ningún valor a la variable II.

En el Logo de Spectrum la equivalente es:

ES.VARIABLE?

Si queremos saber todas las variables que hemos utilizado introducimos, según el ordenador con el que trabajemos:

— PC-COMPATIBLES

? IMVS

Abreviatura de IMprime Variables

En la pantalla aparece la lista de todas las variables utilizadas y sus respectivos contenidos.

— MSX

? LVARs

Abreviatura de Lista de VARIables

Obtenemos en forma de lista exclusivamente el nombre de las variables utilizadas.

— SPECTRUM

? EDVS

Abreviatura de EDita Variables

En el editor aparece la lista de todas las variables utilizadas con sus respectivos contenidos.

En este último caso podemos también

Borra las variables que no necesites.

EXPERIENCIA Y PRACTICAS EN LOGO

elegir la o las variables que queremos que nos muestre:

? EDVS "Nombre de variable

y obtendremos solamente la variable que se especifica y su contenido.

? EDVS [Lista de variables]

mostrará todas las variables que se especifican en la lista y sus respectivos valores.

Si lo que queremos es borrar alguna variable disponemos de la orden:

? BOVAR "Nombre de variable
Abreviatura de BORra VARiable

La variable especificada se borrará de la memoria del ordenador.

Para borrar todas las variables que existan:

? BOVARs

En la versión del Logo del Spectrum las órdenes que realizan estas funciones son:

Para borrar una variable:

? SUPV "Nombre de variable
Abreviatura de SUPrime Variables

Para borrar todas las variables:

? SUPVS

En este caso también podemos borrar un número determinado de variables:

? SUPV [Lista variables]

Se borrarán exclusivamente las variables que se especifican en la lista.

■ Cuadro resumen

— OPERADORES relacionales:

— MAYOR? n1 n2
> n1 n2 forma prefija
n1 > n2 forma infija

Devuelve CIERTO si n1 es mayor que n2; si no, devuelve FALSO.

— MENOR? n1 n2
< n1 n2 forma prefija
n1 < n2 forma infija

Devuelve CIERTO si n1 es menor que n2; si no, devuelve FALSO.

— IGUALES? n1 n2
= n1 n2 forma prefija
n1 = n2 forma infija

Devuelve CIERTO si n1 y n2 son iguales; si no, devuelve FALSO.

— OPERADORES LOGICOS
— NO predicado

Devuelve el valor lógico contrario al del predicado.

— Y predicado1 predicado2

Devuelve CIERTO si todos los datos son ciertos; si no, devuelve FALSO.

— O predicado1 predicado2...

Devuelve FALSO si todos los datos son falsos; si no, devuelve CIERTO.

— VAR? "Nombre de variable
ES.VARIABLE? "Nombre de variable

Devuelve CIERTO si el nombre especificado es el de una variable a la que se la haya asignado un valor. En caso contrario, devuelve FALSO.

— IMVS
LVARs
EDVS

Devuelve una lista de todas las variables utilizadas y sus respectivos contenidos (a excepción de LVARs, que sólo devuelve la lista de los nombres de las variables utilizadas).

— BOVAR "Nombre de variable
SUPV "Nombre de variable

La variable especificada se borrará de la memoria del ordenador.

— BOVARs
SUPVS

Todas las variables existentes serán borradas de la memoria del ordenador.

■ Ejercicios

1. Dibuja un paso de peatones visto desde arriba.

2. Calcula el resultado de la siguiente expresión:

Podemos preguntar si un objeto es una variable.

$\text{O} (Y (20 > (15 * 0.2)) (37 < (90 - 35 * 0.1))) (NO (75 / 6) < 12))$

3. Intenta realizar el siguiente dibujo:

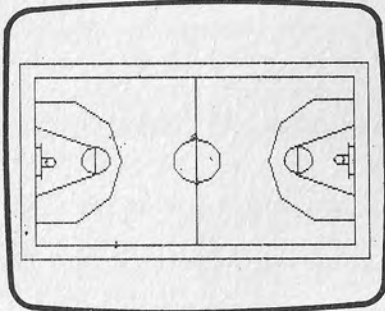


Fig. 1.

4. Tomando como base una caja con las dimensiones que aparecen en la figura:

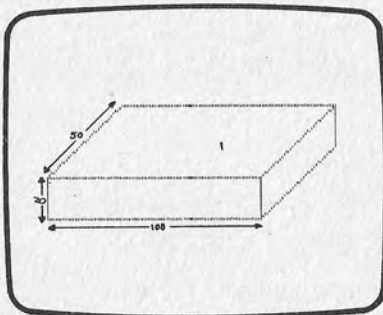


Fig. 2.

define un procedimiento con el cual consigas dividir la caja en departamentos, pudiendo elegir el número de éstos en horizontal y en vertical, para conseguir la caja por ejemplo, de la siguiente forma:

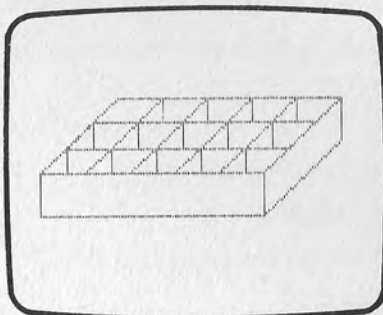


Fig. 3.

5. ¿Son correctas las siguientes órdenes?

- ? ESCRIBE Y CIERTO FALSO
- ? HAZ "A 200
- ? HAZ "B 60
- ? HAZ "C (:A * .5) > (:B * .2)
- ? HAZ "D (:A - 33 * .3) < (:A - :B * .1)
- ? ESCRIBE O :C :D
- ? HAZ "A "CIERTO
- ? ESCRIBE NO :A "CIERTO
- ? HAZ "V 20
- ? ES VAR? "V
- ? BOVAR "V
- ? AV :V

■ Solución a los ejercicios

1:

Definimos un procedimiento de dos entradas, una para elegir la anchura de la calle y otra para elegir el número de franjas.

- :L para la anchura
- :D para el número de franjas
- ? PARA CEBRA :L :D
- > BP
- > PONCL 2
- > SL
- > PONY -80
- > PONX 0
- > BL
- > AV 60
- > SL AV 40
- > BL AV 60
- > HAZ "A 256 - :L
- > HAZ "A :A / 2
- > SL
- > HAZ "IX -128 + :A
- > PONX :IX
- > PONY -80
- > PONRUMBO 0
- > BL
- > AV 160
- > SL
- > PONX :IX + :L
- > BL
- > RE 160
- > SL
- > PONY -20
- > PONX COORX - 5
- > HAZ "S ((:L - 5) - (:D * 5)) / :D
- > PONCL 3

Los operadores de una expresión lógica son predicados.

EXPERIENCIA Y PRACTICAS EN LOGO

```
> REPITE :D [BL REPITE 2 [AV 40 GI 90
AV :S GI 90] SL PONY COORY + 2
PONX COORX - 2 BL RELLENA SL
PONX COORX - (3 + :S) PONY COORY
- 2]
> FIN
```

Ejecútalo, por ejemplo:

? CEBRA 100 10

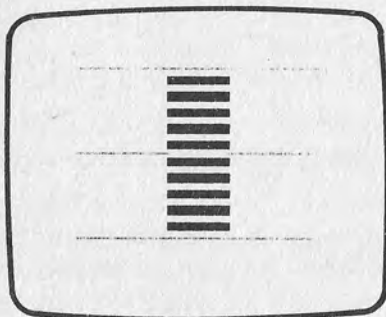


Fig. 4.

o también

? CEBRA 150 15

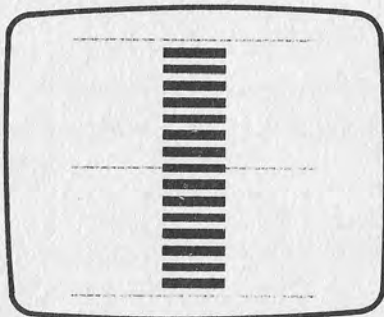


Fig. 5.

Te aconsejamos que antes de ejecutar el procedimiento te asegures de que la división de la longitud de la calle entre el número de franjas es mayor o igual a 7.

• Puedes hacerlo con el LOGO.

```
? ESCRIBE 7 > (150 / 15)
```

CIERTO

150 / 15 = 10

10 es mayor que 7; por tanto, los datos sirven.

2:

La expresión:

```
O ( Y ( 20 > (15 * 0.2) ) ( 37 < (90 - 35
* 0.1) ) ) ( NO ( (75 / 6) > 12 ) )
```

contiene:

Operadores aritméticos: *, -, /
Operadores de relación: >, <
Operadores lógicos: O, Y, NO

Vamos a ir calculando las diferentes operaciones de la expresión.

En primer lugar, nos encontramos con la operación lógica O de otras dos operaciones, una Y u otra NO.

Resolvemos, en primer lugar, la operación lógica Y:

$Y (20 > (15 * 0.2)) (37 < (90 - 35 * 0.1))$

Calculamos el valor de cada uno de los predicados:

$— 20 > (15 * 0.2) = 20 > 3$

Como 20 es mayor que 3, el resultado de esta operación sería:

CIERTO

$— 37 < (90 - 35 * 0.1) = 37 < 86.5$

Como 37 es menor que 86.5, el resultado sería:

CIERTO

Luego la expresión quedaría:

Y "CIERTO "CIERTO

y el resultado sería:

CIERTO

A continuación resolvemos la operación lógica:

$NO ((75 / 6) < 12)$

Calculamos el valor del predicado:

$— (75 / 6) < 12 = 12.5 < 12$

Como 12.5 no es menor que 12, el resultado sería:

FALSO

Luego quedaría:

NO "FALSO

que daría como resultado:

CIERTO

Una vez calculados los predicados de la operación O, sustituimos los valores que hemos obtenido:

O "CIERTO "CIERTO

Las operaciones lógicas también son predicados.

Y el resultado de la expresión general sería:

CIERTO

3:

Para realizar el dibujo del campo de baloncesto hemos definido cuatro procedimientos. El primero, que llamamos CAMPO, dibuja el contorno del campo, la línea de separación y el círculo central:

? PARA A

- > BP
- > SL
- > PONPOS [-120 -50]
- > BL
- > REPITE 2 [AV 140 GD 90 AV 240 GD 90]
- > SL
- > PONPOS [-110 -40]
- > BL
- > REPITE 2 [AV 120 GD 90 AV 220 GD 90]
- > PONX 0
- > AV 120
- > SL
- > PONPOS [-15 15]
- > BL
- > REPITE 10 [AV 10 GD 36]
- > ZONAS
- > FIN

Este procedimiento realiza una llamada al procedimiento ZONAS, que es el que se encarga de dibujar, con la ayuda de otros, las diferentes zonas de un campo de baloncesto, así como las canastas. Para ello va situando a la tortuga en puntos determinados y realizando llamadas a otros procedimientos.

? PARA B

- > SL
- > PONPOS [-110 -5]
- > BL
- > PONPOS [-70 10]
- > SL
- > PONPOS [-110 45]
- > BL
- > PONPOS [-70 30]
- > TIRO
- > SL

- > PONPOS [-110 63]
- > PONRUMBO 90
- > LINEA
- > SL
- > PONPOS [-110 30]
- > PONRUMBO 90
- > CANASTA
- > SL
- > PONPOS [110 -5]
- > BL
- > PONPOS [70 10]
- > SL
- > PONPOS [110 45]
- > BL
- > PONPOS [70 30]
- > TIRO
- > SL
- > PONPOS [110 -23]
- > PONRUMBO -90
- > LINEA
- > SL
- > PONPOS [110 10]
- > PONRUMBO -90
- > CANASTA
- > FIN

Observa que se realizan dos llamadas a tres procedimientos, TIRO, LINEA y CANASTA. Estos dibujan, respectivamente, la zona desde donde se lanzan los tiros libres, la línea de lanzamiento de tres puntos y las canastas. Como esto hay que dibujarlo en las dos partes del campo se hacen dos llamadas a cada uno después de situar a la tortuga en la posición adecuada y con la orientación adecuada.

El procedimiento que dibuja la zona de lanzamiento de tiros libres es:

? PARA TIRO

- > PONRUMBO 90
- > REPITE 20 [AV 3 GD 18]
- > AV 1 GD 90 AV 18
- > FIN

El que dibuja la línea de lanzamiento de tres puntos:

? PARA LINEA

- > BL
- > REPITE 6 [AV 28 GD 36]
- > FIN

Si no estás seguro si has utilizado ya un nombre, para una variable determinada, consulta primero si es así.

EXPERIENCIA Y PRACTICAS EN LOGO

Y, por último, el que dibuja las canastas:

? PARA CANASTA

```
> BL
> AV 4 GI 90 AV 2
> RE 24 AV 2 GI 90
> AV 4 RE 4 GD 90
> AV 6 GD 90 AV 6
> REPITE 30 [AV 1 GI 18]
> AV 7
> FIN
```

Para ejecutarlo basta con introducir el nombre del procedimiento CAMPO:

? CAMPO

4:

En primer lugar, definimos un procedimiento para dibujar la caja:

? PARA CAJA :N :M

```
> BP
> SL
> PONX -50
> BL
> REPITE 2 [AV 20 GD 90 AV 100 GD 90]
> AV 20 GD 45 AV 50
> GD 45 AV 100 GD 135
> AV 50 GI 45 AV 20
> GI 135 AV 50 GI 45
> AV 20
> DPTOS :N :M
> FIN
```

Este procedimiento hace una llamada al procedimiento DPTOS con dos variables, cuyos valores son introducidos desde CAJA y son los que van a determinar el número de departamentos que va a tener la caja en horizontal y en vertical.

El procedimiento DPTOS calcula y dibuja dependiendo de :N y :M los departamentos:

? PARA DPTOS :N :M

```
> SL
> PONPOS [-50 20]
> GD 45
> BL
> HAZ "B 20
```

```
> REPITE :N - 1 [AV 50 / :N GD 45 AV 100 RE 100 GI 90 HAZ "A (COORY - :B) RE :A AV :A HAZ "B COORY GD 45]
> SL
> PONPOS [-50 20]
> BL
> PONRUMBO 90 AV 100 / :M
> HAZ "C 100 / :M
> REPITE :M - 1 [REPITE :N [GI 45 AV 50 / :N GD 135 AV :A RE :A GI 90] SL PONY 20 HAZ "C :C + 100 / :M PONX -50 + :C BL]
> FIN
```

Si ahora lo ejecutamos con los siguientes valores:

? CAJA 2 2

La caja que se obtiene es:

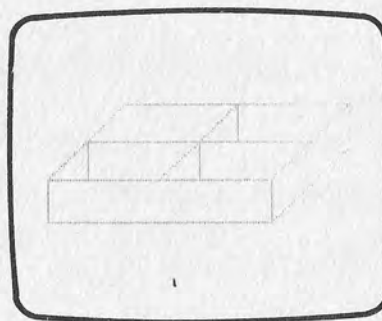


Fig. 6.

Con otros valores:

? CAJA 4 8

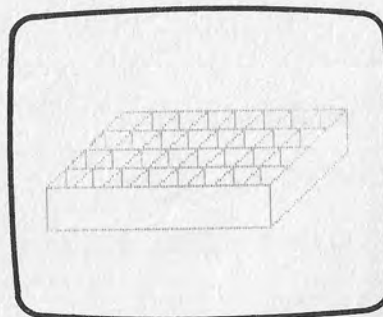


Fig. 7.

El número de los operadores lógicos Y y O puede ser variable.

? CAJA 6 6

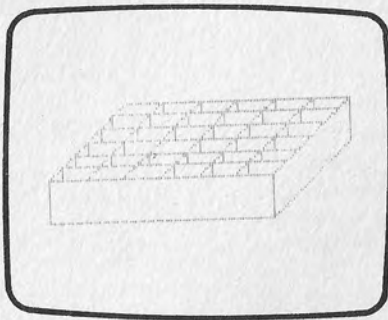


Fig. 8.

5:

— ? ESCRIBE Y CIERTO FALSO

Incorrecta.

Faltan las comillas delante de CIERTO y delante de FALSO.

— ? HAZ "A 200

? HAZ "B 60

? HAZ "C (:A * .5) > (:B * .2)

? HAZ "D (:A - 33 * .3) < (:A - :B * .1)

? ESCRIBE O :C :D

Correcta.

La expresión lógica devuelve CIERTO.

— ? HAZ "A "CIERTO

? ESCRIBE NO :A "CIERTO

Incorrecta.

El operador lógico NO sólo admite un predicado.

El LOGO muestra el mensaje:

NO SE QUE DEBO HACER CON CIER-
TO

— ? HAZ "V 20

? ES VAR? "V

Correcta.

Asignamos a la variable V el valor 20 y a continuación preguntamos si V es variable. Obtendremos como resultado el mensaje CIERTO.

— ? BOVAR "V

? AV :V

Incorrecta.

La orden AV :V no se podrá realizar, ya que anteriormente hemos borrado la variable V.

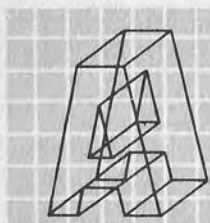
■ ***El operador lógico NO sólo puede tener un dato.***

■ ***Podemos preguntar al logo cuáles son las variables que se utilizan.***

■ ***Las variables pueden ser borradas.***

MANEJO DE SPRITES Y ELEMENTOS GRAFICOS

■ El azar en la animación



la hora de hacer nuestros propios juegos gráficos normalmente, y dependiendo del tipo de juego que queramos realizar, incluiremos una serie de enemigos que nos perseguirán e intentarán matarnos. Tal es el caso de programas como las moscas, los marcianos, el PAC-MAN y muchos otros.

El movimiento de estos personajes habrá de ir encaminado a acercarse lo más posible a nosotros, intentar acorralarnos y matarnos. La forma de realizar el programa para que haga esto depende mucho del juego que queramos realizar. Es por esto que no podemos dar una regla fija que nos indique qué algoritmos tenemos que utilizar.

También nos podemos encontrar programas en los que realmente no es necesario introducir otros personajes que se muevan por pantalla y nos dificulten la terminación del juego. En estos casos el juego suele ser lo sufi-

cientemente complicado como para ponerle más trabas al usuario. Aun así, en programas como el GUSANO LOCO, que ya vimos en tomos anteriores, se podría introducir la figura de un nuevo personaje que se encargase de hacernos la vida imposible. En este caso, dicho personaje no tendría que moverse siguiendo ninguna ley, sino que se movería aleatoriamente por la pantalla con el único propósito de molestarnos.

A continuación vamos a ver un programa en el que el movimiento del enemigo del usuario está realizado al azar. El programa es un juego muy famoso en todos los microordenadores, pero en versión simplificada. Le hemos llamado CARRERA LOCA. El juego consiste en movernos por la pantalla, dejando nuestro rastro, mientras intentamos no salirnos de la pantalla y no chocar contra nosotros mismos. A la vez que nosotros nos movemos, nuestro enemigo se moverá dejando su propio rastro. El objetivo del juego es intentar arrinconar a nuestro oponente de forma que se choque con nosotros, consigo mismo o con los bordes de la pantalla antes de que nosotros lo hagamos.

```
100 REM *****
110 REM * CARRERA LOCA *
120 REM *****
130 REM
140 REM *****
150 REM * INICIALIZACION *
160 REM *****
170 REM
180 CLEAR
190 RANDOMIZE TIMER
200 LET SW=0
210 LET X=5
220 LET X1=5
230 LET X2=35
240 LET X3=35
250 LET Y=12
```

```

260 LET Y1=12
270 LET Y2=12
280 LET Y3=12
290 LET DX=1
300 LET DY=0
310 LET X5=-1
320 LET Y5=0
330 DIM A(40,23)
500 REM
510 REM *****
520 REM * PROGRAMA PRINCIPAL *
530 REM *****
540 REM
550 CLS
560 GOSUB 1000
570 GOSUB 2000
580 GOSUB 3000
590 GOSUB 4000
600 GOSUB 5000
610 GOSUB 6000
620 GOTO 560
1000 REM
1010 REM *****
1020 REM * LECTURA DEL TECLADO *
1030 REM *****
1040 REM
1050 LET A$=INKEY$
1060 IF A$="" THEN RETURN
1070 IF A$="P" OR A$="p" THEN LET DX=1:LET DY=0:RETURN
1080 IF A$="O" OR A$="o" THEN LET DX=-1:LET DY=0:RETURN
1090 IF A$="Q" OR A$="q" THEN LET DX=0:LET DY=-1:RETURN
1100 IF A$="A" OR A$="a" THEN LET DX=0:LET DY=1:RETURN
1110 RETURN
2000 REM
2010 REM *****
2020 REM * COMPROBACION DEL JUGADOR *
2030 REM *****
2040 REM
2050 IF X+DX>40 OR X+DX<1 THEN GOTO 2090
2060 IF Y+DY>22 OR Y+DY<1 THEN GOTO 2090
2070 IF A(X+DX,Y+DY)=1 THEN GOTO 2090
2080 RETURN
2090 CLS
2100 PRINT
2110 PRINT "HAS PERDIDO ...."
2120 PRINT "TE HAS CHOCADO ANTES"
2130 PRINT "QUE TU ENEMIGO"
2140 PRINT
2150 GOTO 7000
3000 REM
3010 REM *****
3020 REM * MOVIMIENTO DEL JUGADOR *
3030 REM *****
3040 REM
3050 LOCATE Y1,X1
3060 PRINT "o"
3070 LET X=X+DX
3080 LET Y=Y+DY
3090 LOCATE Y,X
3100 PRINT "O"
3110 LET Y1=Y
3120 LET X1=X

```

MANEJO DE SPRITES Y ELEMENTOS GRAFICOS

```
3130 LET A(X,Y)=1
3140 RETURN
4000 REM
4010 REM *****
4020 REM * DIRECCION DEL ENEMIGO *
4030 REM *****
4040 REM
4050 IF SW<>5 THEN LET SW=SW+1:RETURN
4060 LET SW=0
4070 LET D=INT(RND*4+1)
4080 IF D=1 THEN LET X5=1:LET Y5=0:RETURN
4090 IF D=2 THEN LET X5=-1:LET Y5=0:RETURN
4100 IF D=3 THEN LET X5=0:LET Y5=-1:RETURN
4110 IF D=4 THEN LET X5=0:LET Y5=1:RETURN
5000 REM
5010 REM *****
5020 REM * COMPROBACION DEL ENEMIGO *
5030 REM *****
5040 REM
5050 IF X3+X5>40 OR X3+X5<1 THEN GOTO 5090
5060 IF Y3+Y5>22 OR Y3+Y5<1 THEN GOTO 5090
5070 IF A(X3+X5,Y3+Y5)=1 THEN GOTO 5090
5080 RETURN
5090 CLS
5100 PRINT "HAS GANADO ...."
5110 PRINT "EL ENEMIGO SE CHOCO"
5120 PRINT "ANTES QUE TU."
5130 PRINT
5140 GOTO 7000
6000 REM
6010 REM *****
6020 REM * MOVIMIENTO DEL ENEMIGO *
6030 REM *****
6040 REM
6050 LOCATE Y2,X2
6060 PRINT "="
6070 LET Y3=Y3+Y5
6080 LET X3=X3+X5
6090 LOCATE Y3,X3
6100 PRINT "*"
6110 LET Y2=Y3
6120 LET X2=X3
6130 LET A(X3,Y3)=1
6140 RETURN
7000 REM
7010 REM *****
7020 REM * OTRA PARTIDA? (S/N) *
7030 REM *****
7040 REM
7050 PRINT
7060 PRINT
7070 PRINT "OTRA PARTIDA? (S/N)"
7080 LET A$=INKEY$
7090 IF A$="" THEN GOTO 7080
7100 IF A$="S" OR A$="s" THEN GOTO 100
7110 IF A$<>"N" AND A$<>"n" THEN GOTO 7080
7120 PRINT
7130 PRINT "ENTONCES ..."
7140 PRINT "          ... ADIOS"
7150 END
```

Las teclas que tendrá que usar el usuario para controlar su coche son las siguientes:

- P — Movimiento hacia la derecha.
- O — Movimiento hacia la izquierda.
- Q — Movimiento hacia arriba.
- A — Movimiento hacia abajo.

Como siempre, a continuación aparecen las modificaciones que hay que realizar en el programa para que pueda funcionar en todos aquellos ordenadores que no son IBM ni compatibles:

COMMODORE:

```
190 LET A=RND (-TI)
550 PRINT CHR$(147)
1050 GET A$
2050 IF X+DX>39 OR X+DX<0 THEN GOTO 2090
2060 IF Y+DY>22 OR Y+DY<0 THEN GOTO 2090
2090 PRINT CHR$(147)
3050 POKE 214, Y1: POKE 211, X1
3090 POKE 214, Y: POKE 211, X
4070 LET D=INT (RND(1)*4+1)
5050 IF X3+X5>39 OR X3+X5<0 THEN GOTO 5090
5060 IF Y3+Y5>22 OR Y3+Y5<0 THEN GOTO 5090
5090 PRINT CHR$(147)
6050 POKE 214, Y2: POKE 211, X2
6090 POKE 214, Y3: POKE 211, X3
7080 GET A$
```

SPECTRUM:

```
190 RANDOMIZE
2050 IF X+DX>31 OR X+DX<0 THEN GOTO 2090
2060 IF Y+DY>21 OR Y+DY<0 THEN GOTO 2090
3050 PRINT AT Y1, X1;
3090 PRINT AT Y, X;
5050 IF X3+X5>31 OR X3+X5<0 THEN GOTO 5090
5060 IF Y3+Y5>21 OR Y3+Y5<0 THEN GOTO 5090
6050 PRINT AT Y2, X2;
6090 PRINT AT Y3, X3;
```

AMSTRAD:

```
190 RANDOMIZE TIME
3050 LOCATE X1, Y1
3090 LOCATE X, Y
6050 LOCATE X2, Y2
6090 LOCATE X3, Y3
```

MSX:

```
190 LET A=RND (-TIME)
3050 LOCATE X1, Y1
3090 LOCATE X, Y
4070 LET D=INT (RND (1)*4+1)
6050 LOCATE X2, Y2
6090 LOCATE X3, Y3
```

Si ejecutamos el programa, podemos ver cómo dos figuras distintas se mueven por la pantalla. Cada una de ellas lleva una trayec-

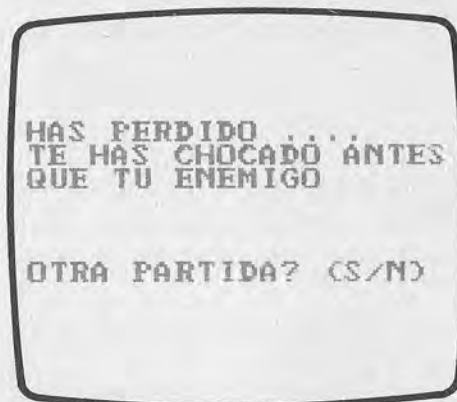


Fig. 1. El programa nos dice cuándo ha habido un choque y quién ha sido el que ha chocado.

toria diferente. La de la derecha lleva la trayectoria que nosotros le vamos indicando. La de la izquierda se mueve de una forma aleatoria por la pantalla. Estamos en un programa que es un ejemplo claro de movimiento simultáneo.

La estructura del programa es la misma que estuvimos estudiando en tomos anteriores. En éstos dijimos que para realizar movimientos simultáneos, como el ordenador no puede hacer dos cosas a la vez, pero es lo suficientemente rápido para que dos cosas que ha hecho seguidas nos parezcan simultáneas, lo mejor era decirle al ordenador que el primero moviese una de las figuras y después la otra. Este programa está realizado de una forma muy modular. Esto nos va a permitir variar cualquiera de las rutinas que lo componen sin influir para nada en el resto del programa. La estructura es la siguiente:

- Lectura del teclado para ver si el jugador ha pulsado alguna tecla.

- Comprobación de que la siguiente posición a la que se va a mover el jugador está permitida. Esto es, mirar si el jugador no se choca con nada en el siguiente movimiento.

- Movimiento del jugador una posición hacia la última dirección indicada por el usuario.

- Cálculo de la siguiente dirección sobre la que se moverá el coche del enemigo.

- Comprobación de que la siguiente posición en pantalla que va a ocupar el enemigo está libre y no se va a chocar con nada.

- Movimiento del enemigo una posición en la dirección que ha calculado el programa.

Esta es la estructura general del corazón del programa. Una vez que se han reali-

EXPERIENCIA Y PRACTICAS EN LOGO

zados todos estos pasos el programa los vuelve a realizar otra vez desde el primero. Esta secuencia se va repitiendo hasta que en alguna de las dos comprobaciones que hay, el ordenador detecta que el jugador o el enemigo ha chocado contra algo. En este momento termina el juego.

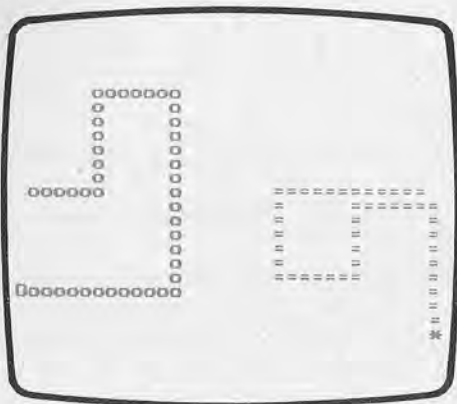


Fig. 2. Un momento en la ejecución del programa.

La explicación del programa es bastante sencilla, pues es muy similar a todo lo que hemos visto hasta ahora. Por ello, sólo daremos una rápida ojeada a su funcionamiento. La única parte del programa a la que dedicaremos más atención es la que se encarga de controlar el movimiento aleatorio del enemigo.

En la línea 180 hacemos un CLEAR. Esto puede parecer un poco tonto, ya que si hacemos RUN esto es como hacer CLEAR:GOTO principio del programa, pero no es así. La primera vez que se ejecuta el programa esta instrucción no sirve para nada, pero a partir de la segunda, una vez que el jugador ha respondido que quiere echar otra partida, la función de esta línea es poner a cero todas las variables y borrar la tabla que se encuentra en la línea 330. Si no pusiésemos esta instrucción, al llegar el programa a la línea 330 nos daría un error por estar definiendo una tabla que ya está definida.

En la línea 190 hacemos que el generador de números aleatorios comience desde un punto desconocido para nosotros. Por ello, utilizamos el valor del tiempo que lleva el ordenador encendido.

Entre las líneas 200 y 330 inicializamos todas las variables que nos van a hacer falta durante el programa. Estas variables, para saber lo que hace cada una de ellas, son:

SW = Se encarga de controlar que el enemigo se ha movido al menos cinco pasos en una misma dirección.

X e Y = Posición en la pantalla de la cabeza de nuestro coche.

X1 e Y1 = Posición anterior de la cabeza de nuestro coche.

X2 e Y2 = Antigua posición de la cabeza de nuestro enemigo.

X3 e Y3 = Posición en la pantalla de la cabeza de nuestro enemigo.

DX = Nos dice si nuestro movimiento se mueve en X y en qué dirección.

DY = Nos dice si nuestro movimiento se mueve en Y y en qué dirección.

X5 = Nos dice si el movimiento del enemigo se mueve en X y en qué dirección.

Y5 = Nos dice si el movimiento del enemigo se mueve el Y y en qué dirección.

A() = Esta tabla será como un espejo de la pantalla y almacenaremos un uno (1) en cada posición que se vaya ocupando en la pantalla.

Entre las líneas 550 y 620 se encuentra el programa principal. Fíjese el lector la comodidad del método y la sencillez con que se entiende todo.

A continuación vienen todas las rutinas, de las que ya hemos hablado y en el mismo orden que hemos visto. La única de ellas que nos interesa en este momento es la que se encarga de decidir en qué dirección será el siguiente movimiento de nuestro enemigo. Esta rutina se encuentra a partir de la línea 4000.

Línea 4050. Preguntamos si la variable SW es distinta de cinco (5). En caso afirmativo hacemos que SW se incremente en uno y devolvemos el control al programa principal. Esta instrucción sirve para que el enemigo dé al menos cinco pasos en una misma dirección antes de cambiar a otra.

Línea 4060. Si SW es igual a cinco, significa que el enemigo ya ha dado cinco pasos en una misma dirección. En este momento podemos poner de nuevo SW a cero para que en el siguiente movimiento siga contando.

Línea 4070. Sacamos un número aleatorio entre 1 y 4. Dependiendo del número que nos salga, el movimiento se realizará en una dirección o en otra.

Línea 4080. Si nos salió un 1, entonces nos moveremos hacia la derecha. Por ello, hacemos que X5 sea igual a 1 e Y5 igual a 0.

Línea 4090. Si salió un 2, entonces nos vamos a mover hacia la izquierda. En este caso X5 es igual a -1 e Y5 igual a 0.

Línea 4100. Si el número aleatorio resulta ser un 3, entonces nos moveremos hacia arriba. X5 será igual a 0 e Y5 igual a -1.

Línea 4110. Por fin, si el número resultante es un 4, entonces nos moveremos hacia abajo, por lo que X5 es igual a 0 e Y5 igual a 1.

Como se puede comprobar al ver la ejecución del programa, este método es un poco rupestre. Si realizamos el programa de esta manera lo único que conseguimos es que el enemigo no dure vivo más de diez segundos. Por ello, sería conveniente modificar esta ru-

tina para que fuese un poco más inteligente, pero sin perder el elemento de azar que la controla. Una posible solución al problema aparece en el programa 2. Lo único que tienes que hacer para introducir este programa en el número uno es borrar las líneas que van desde la 4050 hasta la 4110 e introducir las que aparecen a continuación.

```

4000 REM
4010 REM *****
4020 REM * DIRECCION DEL ENEMIGO *
4030 REM *****
4040 REM
4050 IF X5<>0 THEN IF Y3+Y5>22 OR Y3+Y5<1 OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5
4060 IF Y5<>0 THEN IF X3+X5>40 OR X3+X5<1 OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5
4070 IF SW<>5 THEN LET SW=SW+1:RETURN
4080 LET SW=0
4090 LET D=INT(RND*4+1)
4100 IF (D=1 AND D1=2) OR (D=2 AND D1=1) THEN GOTO 4090
4110 IF (D=3 AND D1=4) OR (D=4 AND D1=3) THEN GOTO 4090
4120 LET D1=D
4130 IF D=1 THEN LET X5=1:LET Y5=0:RETURN
4140 IF D=2 THEN LET X5=-1:LET Y5=0:RETURN
4150 IF D=3 THEN LET X5=0:LET Y5=-1:RETURN
4160 IF D=4 THEN LET X5=0:LET Y5=1:RETURN

```

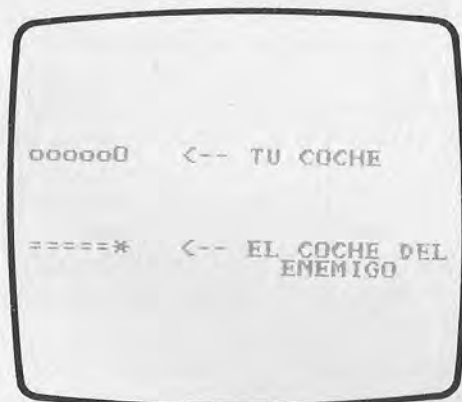


Fig. 3.

Aparte de introducir este grupo de líneas en sustitución de las que había antes, todos los usuarios de ordenadores también tendrán que introducir la siguiente línea al principio del programa:

```
205 LET D1=0
```

Las modificaciones que habría que hacerle a esta rutina para que funcionase en el Commodore y en el Spectrum son las siguientes:

COMMODORE:

```
4050 IF X5<>0 THEN IF Y3+Y5>22 OR Y3+Y5<0
```

```

OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5
4060 IF Y5<>0 THEN IF X3+X5>39 OR X3+X5<0
OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5
4090 LET D=INT(RND(1)*1+1)

```

SPECTRUM:

```

4050 IF X5<>0 THEN IF Y3+Y5>21 OR Y3+Y5<0
OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5
4060 IF Y5<>0 THEN IF X3+X5>31 OR X3+X5<0
OR A(X3+X5,Y3+Y5)=1 THEN LET SW=5

```

Si nos fijamos bien en el listado podemos apreciar que hay muchas cosas que son iguales, pero que también han aparecido algunas líneas distintas. Estas líneas van a realizar una doble comprobación. Por una parte, comprobaremos que en el siguiente movimiento no nos vamos a salir de la pantalla y que no vamos a chocar ni con nosotros mismos ni con el jugador. Por otra parte, y en la segunda comprobación, miramos que la nueva dirección de movimiento no sea contraria a la que llevamos ahora. Esto es, si no estamos moviendo hacia la derecha, que al cambiar de movimiento no nos movamos hacia la izquierda, ya que nos chocaríamos con nosotros mismos.

Se podrían haber introducido más modificaciones en la rutina hasta el punto de que el coche enemigo nunca chocase. Esto tiene

MANEJO DE SPRITES Y ELEMENTOS GRAFICOS

un peligro, y es que si el enemigo queda arrinconado, y no tiene sitio por donde salir, si la rutina es muy perfecta, al ejecutarse esta parte del programa, no se saldría nunca, ya que iría buscando continuamente la posible vía de escape y, al no haberla, el programa no continuaría. Aun así, y para que el lector vaya cogiendo práctica, se le insta para que intente realizar una rutina tan perfecta como ésta.

Ya para terminar vamos a ver cómo funciona esta nueva rutina línea a línea. El lector podrá comprobar que aunque se han introducido líneas de comprobación, la componente de azar no ha sido eliminada.

Línea 4050. Miramos si el movimiento es hacia la derecha o hacia la izquierda. En caso afirmativo, preguntamos si el siguiente movimiento nos va a llevar al borde derecho o izquierdo de la pantalla o si la siguiente posición de nuestro movimiento está ocupada. En caso afirmativo, hacemos que la variable SW sea igual a 5 para, al llegar a la línea 4070, cambiar el sentido del movimiento.

Línea 4060. Miramos si el movimiento es hacia arriba o hacia abajo. En caso afirmativo preguntamos si en el siguiente movimiento nos vamos a chocar con la parte superior o inferior de la pantalla o si la siguiente posición que queremos ocupar ya está ocupada. En caso afirmativo, hacemos que SW sea igual a 5 para cambiar la dirección del movimiento.

Línea 4070. Al igual que vimos en el programa 1, comprobamos si el enemigo ha dado ya cinco pasos. La única diferencia estriba en que esta condición puede ser cierta antes de dar los cinco pasos si se ha variado

el valor de SW en alguna de las dos líneas anteriores para prevenir el choque.

Línea 4080. Hacemos que SW vuelva a valer cero (0).

Línea 4090. Sacamos un número aleatorio entre 1 y 4.

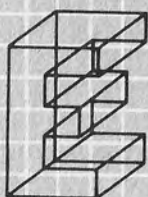
Línea 4100. Preguntamos si el número que hemos sacado con la función RND implica un movimiento de la misma dirección, pero de distinto sentido del que teníamos hace unos instantes. Comprobamos que, si el movimiento es hacia la derecha, el siguiente movimiento no es hacia la izquierda. También comprobamos la inversa. En caso afirmativo, nos volvemos a la línea 4090 para sacar otro número aleatorio.

Línea 4110. Realizamos lo mismo que en la línea anterior, pero esta vez la comprobación es para saber si el movimiento era hacia arriba y ahora es hacia abajo o viceversa. En caso afirmativo nos volvemos a la línea 4090 para sacar otro número aleatorio. Gracias a este grupo de líneas, conseguimos que el coche no dé una vuelta de 180 grados para chocarse consigo mismo.

Línea 4120. Una vez que hemos hecho todas las comprobaciones que queríamos, almacenamos en D1 el valor de D. Esto nos sirve para comprobar en la siguiente ocasión que lo necesitamos que el movimiento no cambia sólo de sentido.

A partir de la línea 4130 y hasta la 4160, el funcionamiento del programa es igual que lo que hemos explicado anteriormente. Por ello, no es necesario repetir la explicación.

TRUCOS Y RUTINAS BASICAS



En tomos anteriores vimos cómo realizar un programa que gestionase nuestros ficheros. Una parte muy importante de este programa era la ordenación de las fichas. Para ello utilizamos una de las muchas rutinas que se pueden realizar. En este tomo y en los siguientes veremos algunas rutinas de ordenación (también llamadas SORTs). Algunas serán numéricas, otras alfanuméricas, las habrá mixtas, más rápidas y más lentas, más complicadas y más sencillas.

Antes de ver la primera de las rutinas, proponemos un programa que nos permitirá ejecutar dichas rutinas una a una. Con este programa podremos introducir por el teclado los distintos datos a ordenar, llamar a la rutina e imprimirlos, una vez ordenados en la pantalla. Este programa, por supuesto, no es más que un programa ejemplo, y el lector tendrá que realizar su propio programa que gestione la rutina de ordenación que él desee con más precisión y mejor.

```
100 REM *****
110 REM * PROGRAMA DEMO PARA EJECUTAR LOS PROGRAMAS *
120 REM * 2, 3, 4 Y 5. *
130 REM *****
140 REM
150 CLS
160 INPUT "NUMERO DE ELEMENTOS A ORDENAR ";NU
170 CLS
180 DIM T$(NU)
190 PRINT "INTRODUZCA LOS ELEMENTOS A ORDENAR UNO A UNO"
200 FOR I=1 TO NU
210   PRINT "ELEMENTO No. ";I;
220   INPUT T$(I)
230 NEXT I
240 CLS
250 PRINT "ESTOS SON LOS ELEMENTOS A ORDENAR"
260 PRINT
270 FOR I=1 TO NU
280   PRINT T$(I)
290 NEXT I
300 PRINT
310 PRINT "ORDENANDO. ESPERE UN MOMENTO"
320 GOSUB 1000
330 CLS
340 PRINT "ESTOS SON LOS ELEMENTOS ORDENADOS"
350 PRINT
360 FOR I=1 TO NU
370   PRINT T$(I)
380 NEXT I
390 END
```

TRUCOS Y RUTINAS BASICAS

Las modificaciones que hay que realizar en este programa son muy pocas y sólo para el COMMODORE y SPECTRUM. Estas son:

COMMODORE:

```
150 PRINT CHR$(147)
170 PRINT CHR$(147)
240 PRINT CHR$(147)
330 PRINT CHR$(147)
```

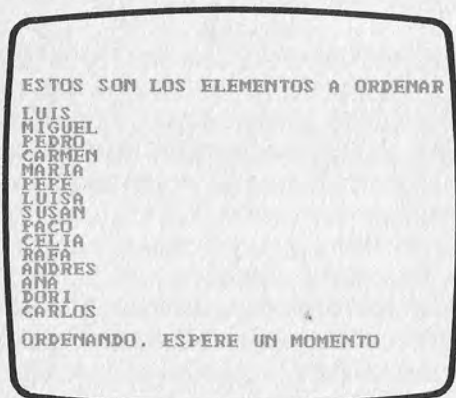


Fig. 1. Lista de nombres a ordenar antes de llamar a cualquiera de estas rutinas.

SPECTRUM:

Sólo hay que cambiar la línea 180 por:
180 DIN T\$(NU,30)

El segundo número dentro de los paréntesis nos dice el número máximo de caracteres que puede tener cada elemento a ordenar. Si el usuario quiere ordenar cadenas de caracteres que tengan más de 30 caracteres (o menos), tendrá que cambiar dicho número por el que le interese.

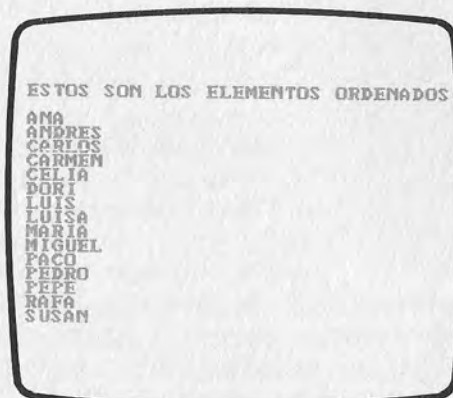


Fig. 2. La misma lista de nombres una vez ordenada.

A continuación vamos a ver la primera de las rutinas de ordenación. Esta rutina emplea el método llamado de burbuja o binario.

Es el ejemplo más sencillo de ordenación. Este tipo de ordenación fue el que utilizamos en el programa gestor de ficheros.

```
1000 REM *****
1010 REM *
1020 REM * <<< ORDENACION NUM.POR EL METODO DE BURBUJA CON MARCA >>> *
1030 REM *
1040 REM * VARIABLES QUE HAY QUE PASARLE AL PROGRAMA
1050 REM * -----
1060 REM *
1070 REM * NU = NUMERO DE ELEMENTOS A ORDENAR
1080 REM * T() = TABLA CON LOS ELEMENTOS A ORDENAR
1090 REM *
1100 REM * VARIABLES INTERNAS USADAS POR LA RUTINA
1110 REM * -----
1120 REM *
1130 REM * A = VARIABLE AUXILIAR
1140 REM * I = VARIABLE INDICE DEL PRIMER BUCLE
1150 REM * J = VARIABLE INDICE DEL SEGUNDO BUCLE
1160 REM * SW = SWICH INDICADOR DE LA ORDENACION
1170 REM *
1180 REM * LA SUBROUTINA NOS RETORNA CON LOS DATOS
```

```

1190 REM * ORDENADOS EN T(). *
1200 REM * *
1210 REM *****
1220 REM
1230 LET SW=0
1240 FOR I=1 TO NU-1
1250 IF T(I+1)<T(I) THEN LET SW=1:LET A=T(I+1):LET T(I+1)=T(I):LET T(I)=A
1260 NEXT I
1270 IF SW=1 THEN GOTO 1230
1280 RETURN

```

El programa no necesita ninguna modificación para funcionar. Puede funcionar tal y como aparece en todos los ordenadores.

Se recuerda al lector que para que el programa funcione tiene que tener definida la variable alfanumérica T\$(), así como el número de elementos a ordenar (NU). Para ello se puede utilizar el programa 1 que hemos visto un poco más arriba. Lo único que hay que hacer es unir los dos en un mismo listado.

El funcionamiento del programa es muy sencillo. El método consiste en comparar el primer elemento a ordenar con todos los demás. Cada vez que nos encontremos uno que

es menor que éste, intercambiaremos sus posiciones, con lo que el nuevo elemento pasará a ser el menor y a colocarse en la cabeza de la lista. Una vez que nos hemos recorrido la lista entera y hemos puesto el más pequeño a la cabeza de ésta, realizamos la misma operación, pero pensando en la segunda posición de la lista. Este proceso lo repetimos hasta que hemos llegado hasta el último elemento de la lista. En ese momento la lista con todos los elementos estará ordenada.

El siguiente programa, que tampoco necesita de ninguna variación, utiliza un método muy parecido al anterior, pero en ocasiones puede resultar más rápido.

```

1000 REM *****
1010 REM * *
1020 REM * <<< ORDENACION POR EL METODO DE BURBUJA CON MARCA >>> *
1030 REM * *
1040 REM * VARIABLES QUE HAY QUE PASARLE AL PROGRAMA *
1050 REM * ----- *
1060 REM * *
1070 REM * NU = NUMERO DE ELEMENTOS A ORDENAR *
1080 REM * T$( ) = TABLA CON LOS ELEMENTOS A ORDENAR *
1090 REM * *
1100 REM * VARIABLES INTERNAS USADAS POR LA RUTINA *
1110 REM * ----- *
1120 REM * *
1130 REM * A$ = VARIABLE AUXILIAR *
1140 REM * I = VARIABLE INDICE DEL PRIMER BUCLE *
1150 REM * J = VARIABLE INDICE DEL SEGUNDO BUCLE *
1160 REM * SW = SWICH INDICADOR DE LA ORDENACION *
1170 REM * *
1180 REM * LA SUBROUTINA NOS RETORNA CON LOS DATOS *
1190 REM * ORDENADOS EN T$( ). *
1200 REM * *
1210 REM *****
1220 REM
1230 LET SW=0
1240 FOR I=1 TO NU-1
1250 IF T$(I+1)<T$(I) THEN LET SW=1:LET A$=T$(I+1):LET T$(I+1)=T$(I):LET T$(I)=A$
1260 NEXT I
1270 IF SW=1 THEN GOTO 1230
1280 RETURN

```

TRUCOS Y RUTINAS BASICAS

El defecto del programa 2 radicaba en que si la lista a ordenar estaba ya ordenada, la rutina no se daba cuenta de ello y realizaba todas las comparaciones pertinentes. Este nuevo programa sí realiza dicha comparación y, en caso de que la lista esté ordenada, vuelve al programa principal.

Este segundo programa puede parecer un poco inútil, pero nada más lejos de la realidad. Hay veces que, al realizar nuestros propios programas, si trabajamos con un fichero, éste se ordena automáticamente al final de la sesión. Como puede darse el caso de que hayamos ejecutado el programa sin realizar ningún cambio en los registros que gestiona (por ejemplo, si lo hemos utilizado sólo para listar), si el programa se ordena automáticamente al terminar la sesión, es preferible que la rutina de ordenación detecte si el fichero se encuentra ya ordenado.

La forma de comprobar que dicho fichero está ordenado es definir una variable

SWICH (como un interruptor) con valor cero. Si dentro de la primera vuelta tenemos que cambiar dos elementos de lugar, eso significará que el fichero no estaba ordenado y entonces hacemos que dicha variable tome el valor uno. En el caso de que no hayamos tenido que cambiar ningún elemento de sitio, como SW seguirá valiendo cero, gracias a la comprobación de la línea 1270 retornaremos al programa principal.

Si SW tenía valor uno nos volvemos a la línea 1230 para realizar una vuelta más. En ese momento SW vuelve a coger valor cero. Con esto se consigue que si la lista se ordena de varias pasadas, pero no de todas, se detectará y se saldrá de la rutina, con lo que el tiempo de ejecución de ésta puede ser menor que el del programa visto anteriormente.

La tercera rutina que vamos a ver utiliza un método algo distinto y más complicado para ordenar la lista de números. Este tipo de ordenación se llama búsqueda del máximo y del mínimo.

```

1000 REM *****
1010 REM *
1020 REM * <<< ORDENACION POR EL METODO DE BUSQUEDA MAX Y MIN >>> *
1030 REM *
1040 REM * VARIABLES QUE HAY QUE PASARLE AL PROGRAMA
1050 REM * -----
1060 REM *
1070 REM * NU = NUMERO DE ELEMENTOS A ORDENAR
1080 REM * T$( ) = TABLA CON LOS ELEMENTOS A ORDENAR
1090 REM *
1100 REM * VARIABLES INTERNAS USADAS POR LA Rutina
1110 REM * -----
1120 REM *
1130 REM * I$ = VARIABLE AUXILIAR
1140 REM * I = VARIABLE INDICE DEL PRIMER BUCLE
1150 REM * J = VARIABLE INDICE DEL SEGUNDO BUCLE
1160 REM * SW = SWICH INDICADOR DE LA ORDENACION
1170 REM * M$ = VARIABLE CON EL ELEMENTO MAS ALTO
1180 REM * M = VARIABLE POSICION ELEMENTO MAS ALTO
1190 REM * W$ = VARIABLE CON EL ELEMENTO MAS BAJO
1200 REM * W = VARIABLE POSICION ELEMENTO MAS BAJO
1210 REM * N = VARIABLE AUXILIAR
1220 REM *
1230 REM * LA SUBROUTINA NOS RETORNA CON LOS DATOS
1240 REM * ORDENADOS EN T$( ).
1250 REM *
1260 REM *****
1270 REM
1280 FOR I=1 TO INT(NU/2)
1290 LET M$=CHR$(255)
1300 LET W$=CHR$(0)
1310 LET N=NU-I+1
1320 FOR J=I TO N
1330 IF T$(J)<M$ THEN LET M$=T$(J):LET M=J
1340 IF T$(J)>W$ THEN LET W$=T$(J):LET W=J
1350 NEXT J
1360 LET I$=T$(N)

```

```

1370 LET T$(M)=T$(I)
1380 LET T$(W)=T$(N)
1390 LET T$(I)=M$
1400 LET T$(N)=W$
1410 IF W=I AND M<>N THEN LET T$(M)=I$
1420 NEXT I
1430 RETURN

```

Este método de ordenación consiste en encontrar el mayor y el menor elemento de la lista y colocarlos en la primera y en la última posición, respectivamente. A continuación se buscan los dos siguientes por arriba (el siguiente más mayor) y por abajo (el siguiente menor) y se ponen en la segunda y penúltima posición de la tabla. Este proceso se va repitiendo hasta que toda la lista esté ordenada. Lo que tiene de bueno este sistema es que coloca dos elementos de la lista cada vez que tiene que hacerlo; con ello se consigue que el programa dé la mitad de pasadas que los anteriores. El único problema que tiene es que no nos permite comprobar si el fichero estaba clasificado con antelación.

Como este método es algo más complicado y desconocido, vamos a explicar su funcionamiento línea a línea:

Línea 1280. Comenzamos un bucle desde (1) hasta la mitad de la lista de elementos a ordenar. Esto es así porque como en cada paso se organizan dos de los elementos, sólo hay que hacer la comprobación la mitad de las veces.

Línea 1290. Hacemos que M\$ sea igual al carácter 255. Esta variable se encargará de ir almacenando el elemento menor de la lista. En un principio se asigna el carácter 255 porque no puede haber ningún elemento con un código ASCII mayor que éste.

Línea 1300. Hacemos que W\$ sea igual al carácter número 0. Esta variable será la encargada de almacenar el mayor elemento de la lista. Al principio toma el valor del carácter 0, porque no puede haber un elemento de la lista con un código ASCII menor.

Línea 1310. Hacemos que la variable N tome el valor del último elemento de la lista. En la primera vuelta del bucle tenemos que I es igual a uno (1) y que N es igual a NU. En la segunda tenemos que I es igual a dos (2) y N a NU-1 y así sucesivamente. Según se puede apreciar, después de cada vuelta, estas dos variables se van moviendo hacia el centro de la lista. Una va aumentando, mientras la otra va disminuyendo.

Línea 1320. Comenzamos un nuevo bu-

cle que va desde I hasta N. Esto está hecho así para que dicho bucle en la primera vuelta coja los elementos que van desde el primero al último (NU). La segunda vez que se ejecute irá desde el segundo al penúltimo (NU-1) y así hasta el final. Con esto se consigue que los elementos que ya han sido ordenados no sean tocados en las restantes vueltas del bucle.

Línea 1330. Comprobamos si el carácter que estamos apuntando con la variable J es menor que M\$. En caso afirmativo hacemos que M\$ tome dicho valor y almacenamos en la variable M la posición de este elemento dentro de la lista.

Línea 1340. Comprobamos si el elemento que nos toca comparar es mayor que W\$. Si es así, entonces hacemos que W\$ tome el valor de este elemento y almacenamos su posición dentro de la lista en W.

Línea 1350. Aquí termina el segundo bucle. En el correr de este bucle, y cuando termina, nos encontramos que, gracias a las líneas 1330 y 1340, tenemos almacenado en M\$ el menor de los elementos entre los límites de la lista impuestos por I y N. Lo mismo ocurre con W\$, en esta variable queda almacenado el mayor de todos. Las posiciones que éstos ocupaban anteriormente estarán almacenadas en las variables numéricas M y W.

Línea 1360. Almacenamos en la variable alfanumérica I\$ el valor de T\$(n). T\$(n) es el último elemento del grupo de la lista que estamos ordenando.

Línea 1370. Almacenamos en T\$(m) el valor de T\$(i). Con esto lo que hacemos es cambiar el lugar dentro de la lista del primer elemento de ésta.

Línea 1380. Almacenamos en T\$(w) el valor de T\$(n). Lo mismo que en la línea anterior, pero con el último elemento de la lista.

Línea 1390. Almacenamos en T\$(i) el valor de M\$. Hay que recordar que M\$ tiene almacenado el elemento más bajo de la lista.

Línea 1400. Almacenamos en T\$(n) el valor de W\$. W\$ tiene almacenado el mayor elemento de la lista. Estas cuatro últimas líneas, como bien se puede apreciar, son las encargadas de intercambiar ciertos elementos

TRUCOS Y RUTINAS BASICAS

de la tabla entre sí. Se colocan los elementos mayor y menor en los extremos de la lista y los elementos que antes había en dichos extremos se colocan en las posiciones que antes ocupaban los elementos mayor y menor.

Línea 1420. Aquí termina el bucle.

Línea 1430. Devolvemos el control al programa inicial que llamó a esta rutina.

El último tipo de ordenación (SORT) que vamos a ver en este tomo es el llamado de intercambio retardado. En realidad, es una variante del de burbuja, pero en ocasiones puede ordenarnos una lista de elementos mucho más rápido.

```

1000 REM *****
1010 REM *
1020 REM * <<< ORDENACION POR EL METODO DEL INTERCAMBIO RET. >>> *
1030 REM *
1040 REM * VARIABLES QUE HAY QUE PASARLE AL PROGRAMA
1050 REM * -----
1060 REM *
1070 REM * NU = NUMERO DE ELEMENTOS A ORDENAR
1080 REM * T$( ) = TABLA CON LOS ELEMENTOS A ORDENAR
1090 REM *
1100 REM * VARIABLES INTERNAS USADAS POR LA RUTINA
1110 REM * -----
1120 REM *
1130 REM * A$ = VARIABLE AUXILIAR
1140 REM * I = VARIABLE INDICE DEL PRIMER BUCLE
1150 REM * J = VARIABLE INDICE DEL SEGUNDO BUCLE
1160 REM * V = VARIABLE AUXILIAR
1170 REM *
1180 REM * LA SUBROUTINA NOS RETORNA CON LOS DATOS
1190 REM * ORDENADOS EN T$( ).
1200 REM *
1210 REM *****
1220 REM
1230 FOR I=1 TO NU-1
1240 LET V=I
1250 FOR J=I+1 TO NU
1260 IF T$(J)<T$(V) THEN LET V=J
1270 NEXT J
1280 IF V<>I THEN LET A$=T$(I):LET T$(I)=T$(V):LET T$(V)=A$
1290 NEXT I
1300 RETURN

```

Esta nueva rutina de ordenación es más rápida siempre que no tengamos que ordenar una serie muy grande de elementos. El método de ordenación se llama por intercambio, porque vamos intercambiando las posiciones de los elementos hasta que toda la lista está ordenada. En esto la rutina funciona igual que la anteriormente vista de burbuja. Se denomina retardado porque en cada pasada por la lista no se realiza ningún cambio de posición hasta que no hemos encontrado el menor elemento de ésta.

Para saber cuál es el menor elemento de la lista utilizamos una variable numérica a modo de puntero (V). Esta variable irá apuntando al elemento más pequeño de la lista. Cada vez que nos encontremos con un ele-

mento más pequeño que al que apunta el puntero, cambiaremos el valor del puntero.

Una vez que hemos terminado de observar y comparar toda la lista, colocamos el elemento al cual apunta el puntero en la primera posición, el elemento que estaba en la primera posición en la posición a la que apunta el puntero y continuamos con la siguiente vuelta del bucle partiendo desde el siguiente elemento.

NOTA: Hay que tener en cuenta que estas rutinas están preparadas para ordenaciones alfanuméricas. Esto significa que pueden ordenar lo mismo letras que números, que letras y números. Pero, si el lector necesita ordenar solamente números, se le recomienda

que cambie toda las variables alfanuméricas por numéricas para aumentar la velocidad de ordenación del SORT en cuestión. Un ejemplo

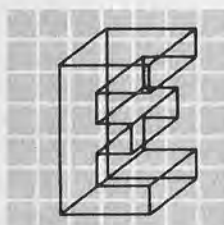
de cómo quedaría el programa 3 preparado para ordenar sólo series de números es el que aparece como programa 6.

```

1000 REM *****
1010 REM *
1020 REM * <<< ORDENACION POR EL METODO DE BURBUJA >>> *
1030 REM *
1040 REM * VARIABLES QUE HAY QUE PASARLE AL PROGRAMA *
1050 REM * ----- *
1060 REM *
1070 REM * NU      = NUMERO DE ELEMENTOS A ORDENAR *
1080 REM * T$( ) = TABLA CON LOS ELEMENTOS A ORDENAR *
1090 REM *
1100 REM * VARIABLES INTERNAS USADAS POR LA RUTINA *
1110 REM * ----- *
1120 REM *
1130 REM * A$      = VARIABLE AUXILIAR *
1140 REM * I       = VARIABLE INDICE DEL PRIMER BUCLE *
1150 REM * J       = VARIABLE INDICE DEL SEGUNDO BUCLE *
1160 REM *
1170 REM * LA SUBROUTINA NOS RETORNA CON LOS DATOS *
1180 REM * ORDENADOS EN T$( ). *
1190 REM *
1200 REM *****
1210 REM
1220 FOR I=1 TO NU-1
1230   FOR J=I+1 TO NU
1240     IF T$(I)>T$(J) THEN LET A$=T$(I):LET T$(I)=T$(J):LET T$(J)=A$
1250   NEXT J
1260 NEXT I
1270 RETURN

```

ANÁLISIS DE SEÑALES ACÚSTICAS



En un tomo anterior hemos presentado la posibilidad de hacer hablar al ordenador mediante un circuito de síntesis utilizando segmentos elementales de voz que hemos denominado alófonos. Como primer paso para

comprender el proceso de análisis de voz, vamos a describir circuitos que nos permitan descomponer las señales acústicas en general y las relacionadas con la voz en particular en sus componentes espectrales y su variación con el tiempo, para poder conocerlas a fondo.

Con un adaptador para adquisición de datos analógicos, como el descrito en uno de los tomos anteriores, podríamos adquirir la señal eléctrica resultante de la generada por un micrófono y almacenarla en memoria para su tratamiento. Mediante programas apropiados podríamos determinar los parámetros fundamentales que definen la señal: variación de la amplitud, espectro de frecuencias, tono fundamental de la voz, formantes, etc. El problema puede resultar complejo si consideramos la cantidad de memoria necesaria para almacenar una determinada longitud de tiempo de evolución de la señal.

Si nos limitamos al estudio de señales producidas por la voz, hemos de definir la frecuencia de muestreo mínima para adquirir la señal con toda la resolución necesaria. La voz humana posee frecuencias comprendidas entre unos 50 y 5.000 Hz. La frecuencia inferior corresponde a la voz de hombre de tono muy bajo, lo normal es que ronde los 200 Hz. La frecuencia superior corresponde a los armónicos más altos de la voz de mujer y niños. La banda pasante necesaria para poder obtener una calidad aceptable para reconocer la voz de una persona es de 300 a 3.400 Hz, que es el ancho de banda de los circuitos utilizados nor-

malmente en telefonía. Si se dispone de mayor ancho de banda, la calidad es mayor y pueden percibirse mayores detalles de entonación, acento y demás características peculiares de cada voz.

De acuerdo con la fórmula de Shannon sobre el muestreo, ya mencionada en el tomo de adquisición de datos analógicos, el número de muestras debe ser superior al doble de la frecuencia del armónico más alto existente en la señal. Para la digitalización de voz sería necesario por lo menos 6.800 muestras por segundo. En telefonía digital se utiliza normalmente muestreo a 8.000 muestras por segundo, que cubre el margen necesario con creces.

En cuanto a la resolución, es interesante disponer del mayor número de niveles posible, para poder admitir un amplio margen de intensidades de voz. En la telefonía digital se utiliza una compresión previa de la señal, que consiste en aplicar una etapa no lineal, de forma que se realza la codificación de los niveles bajos, con lo que se mejora la respuesta real del sistema para un número determinado de bits de codificación. El número de niveles usual es de 256, correspondientes a 8 bits.

Sin embargo, vamos a contemplar una solución más práctica, aunque menos precisa, para representar la voz y la evolución de las señales acústicas, como introducción al tratamiento de las señales.

Si recordamos uno de los métodos ideados para el análisis y síntesis de voz, denominado VOCODER, descrito en el tomo anterior, vemos que su esquema de bloques se compone de un amplificador de entrada y de un conjunto de filtros en paralelo, para diferentes frecuencias del espectro audible. Las amplitudes captadas por estos amplificadores y su evolución en el tiempo, describen la señal de una forma muy gráfica y palpable. Utilizando el or-

denador, podríamos incluso analizar y almacenar los valores correspondientes a las frecuencias para determinadas voces y utilizarlo para poder controlar nuestro ordenador directamente con la voz. Debemos diseñar un circuito que permita este tipo de captación de señal y el programa correspondiente que lo haga efectivo y utilizable.

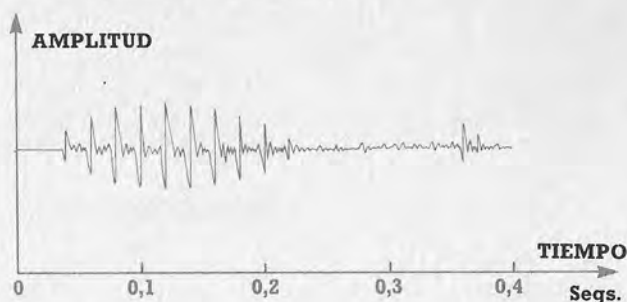


Fig. 1. Señal de voz.

Es interesante observar la forma de onda de una señal generada por un micrófono a partir de la voz. Lo que nos muestra el osciloscopio es la amplitud de la onda de presión producida por el aire en las cuerdas bucales y modulado por la boca. Se presentan dos tipos de ondas:

- Ondas de apariencia repetitiva, más o menos veces y de amplitud bastante grande. Corresponden a la pronunciación vocálica. Son las sílabas sonoras.

- Ondas de apariencia aleatoria, de amplitud muy baja. Corresponden a la pronunciación de consonantes.

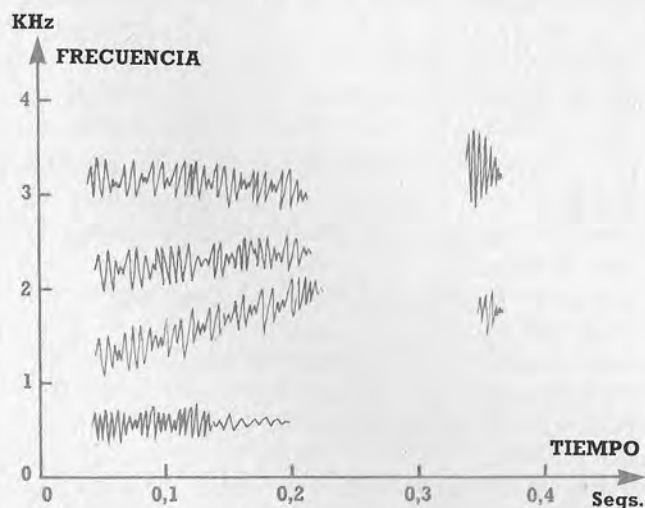


Fig. 2. Espectro de la señal.

La observación de la misma señal, pero contemplando el espectro de frecuencias emitidas, requiere un equipo más costoso y no fácilmente disponible: un analizador de espectro en tiempo real.

Vamos a ver que con algunos circuitos y nuestro ordenador personal, podemos generar un diagrama muy parecido al espectro en tiempo real. Generalmente sería necesario disponer de algún circuito que hiciera el cálculo del espectro de forma rápida, mediante la técnica de la transformada rápida de Fourier, pero vamos a ver que un poco de hardware puede resolverse el problema de una manera más eficiente.

Diagrama de bloques

Aunque pueden ofrecerse diferentes soluciones para resolver el problema que nos planteamos, vamos a limitarnos a una solución práctica y efectiva, que ya ampliaremos, si seguimos investigando en el tema.

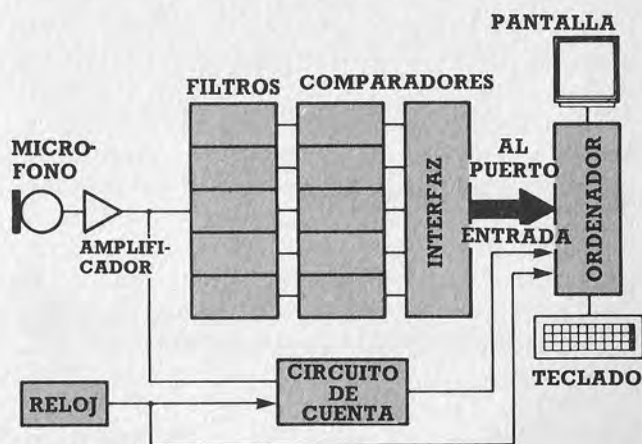


Fig. 3. Diagrama de bloques.

En el diagrama se muestran los bloques principales del circuito:

- Micrófono.
- Amplificador de entrada, con control de nivel.
- Filtros de cada banda.
- Comparadores.
- Interfaz con el ordenador.
- Reloj.
- Circuito de cuenta de pasos por cero.
- El ordenador personal.

Amplificador de entrada

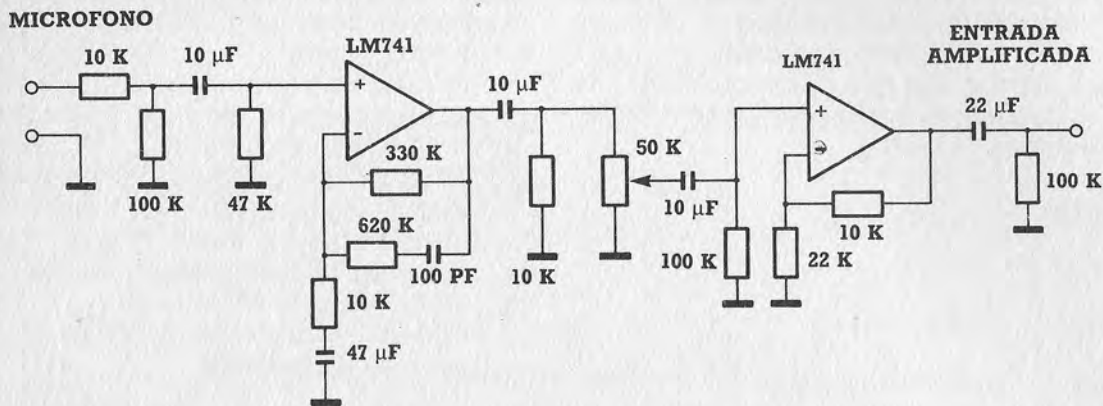


Fig. 4. Amplificador de entrada.

Empezamos amplificando la señal, para adaptar los niveles propios del micrófono a los niveles necesarios para el proceso en los filtros. La ganancia se fija mediante un potenciómetro y deberá ajustarse en cada caso según el equipo de que dispongamos. El amplificador también filtra las frecuencias superiores e inferiores para reducir en lo posible los errores de falsa interpretación de frecuencia.

Indicador de nivel.

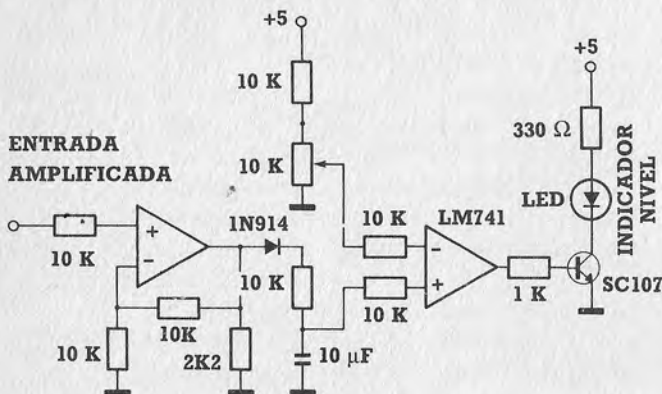


Fig. 5. Indicador de nivel.

Conviene incluir en la salida del amplificador un indicador de nivel, para controlar visualmente cuando estamos dando señal de amplitud suficiente. La salida del indicador de nivel podría utilizarse como medida de la energía de la señal. Por supuesto, que para poder ser utilizada por el ordenador sería nece-

sario realizar la conversión analógico/digital. La señal se compara con un nivel, ajustable mediante un potenciómetro, y si supera un determinado umbral hace lucir un diodo LED.

Filtros de banda

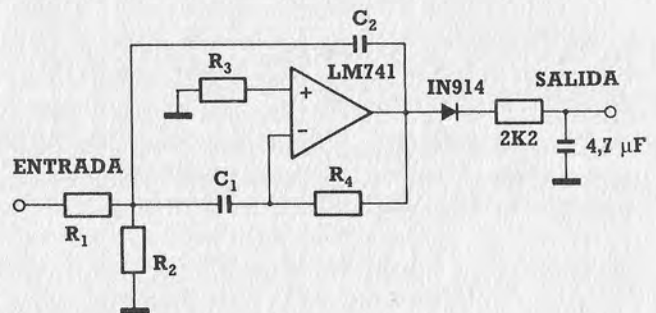


Fig. 6. Circuitos pasa-banda.

Los filtros de cada banda están diseñados mediante amplificadores operacionales en configuración pasabanda. No necesitan ser de una frecuencia muy exacta ni de un factor de calidad elevado, solamente es necesario que se puedan distinguir las frecuencias en saltos separados aproximadamente una octava cada uno. Las constantes de tiempo para la fijación de las frecuencias de corte vienen fijadas por los valores de las resistencias R1, R2, R3 y R4 y los condensadores C1 y C2. En el esquema completo se indican los valores para cada banda, así como en la tabla adjunta:

Tabla 1 Valores de resistencias y condensadores							
Frecuencia	R1	R2	R3	R4	R5	C1	C2
30 Hz	43K	82K	750K	430K	2K2	47nF	47nF
60 Hz	22K	39K	390K	220K	2K2	47nF	47nF
120 Hz	11K	22K	220K	110K	2K2	47nF	47nF
250 Hz	5K6	10K	110K	56K	2K2	47nF	47nF
500 Hz	3K9	6K8	56K	39K	2K2	33nF	33nF
1.000 Hz	6K2	12K	27K	62K	2K2	10nF	10nF
2.000 Hz	4K7	8K2	15K	47K	2K2	6nF8	6nF8
4.000 Hz	3K3	6K8	7K5	33K	2K2	4nF7	4nF7

Comparadores

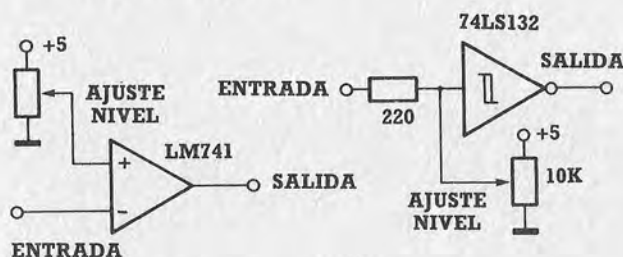


Fig. 7. Circuitos comparadores.

La salida de cada filtro es rectificada y filtrada para dar una señal proporcional a la amplitud de la señal en la banda correspondiente. La señal de amplitud se compara con un nivel de referencia y se genera un nivel lógico indicativo de estar por encima o por debajo de un umbral. En realidad, deberíamos medir el valor del nivel con al menos 6 bits de precisión, pero para este experimento es suficiente con obtener la indicación de que la señal es superior o inferior a un nivel determinado.

Puede hacerse una simplificación de la comparación mediante circuitos inversores con histéresis del tipo 74LS14 o 74LS132, aunque el resultado no será tan preciso.

Reloj y circuito de muestreo

Las señales indicadoras de nivel en cada uno de los filtros deben ser muestreadas periódicamente para tener una indicación temporal de la ocurrencia de cada una de ellas. La lectura debería sincronizarse con esta señal de muestreo. Podría utilizarse el reloj interno del ordenador para determinar cuándo se realizan las muestras, pero es preferible generarlo externamente para poderlo

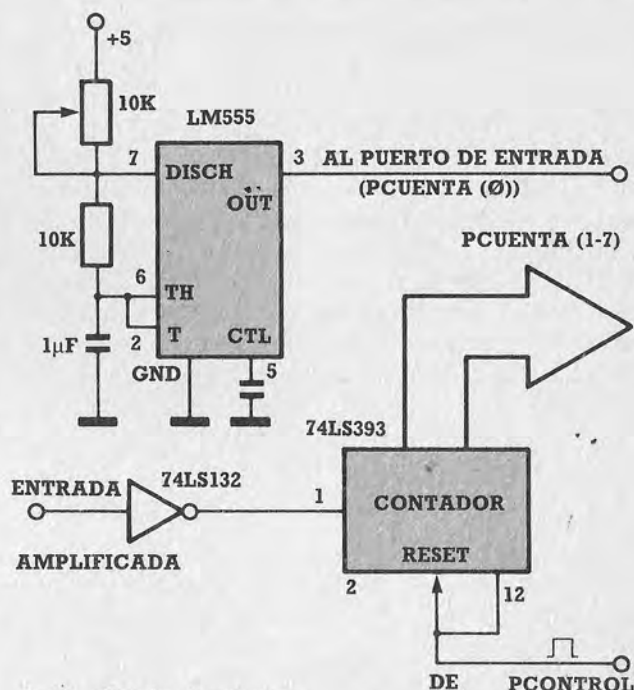


Fig. 8. Circuito de muestreo.

ajustar de manera manual. El circuito empleado es el LM555 en configuración astable, ajustable mediante una resistencia variable. En cualquier caso, siempre podremos activarlo por programa si así se desea. El circuito incluye la medida del número de pasos por cero de la señal, que se emplea como indicación grosera de la frecuencia media en el intervalo. Puede considerarse con bastante aproximación que es un valor proporcional al tono de la señal. El contador de pasos por cero debe ser puesto a cero después de cada lectura del valor alcanzado en el intervalo anterior.

Interfaz

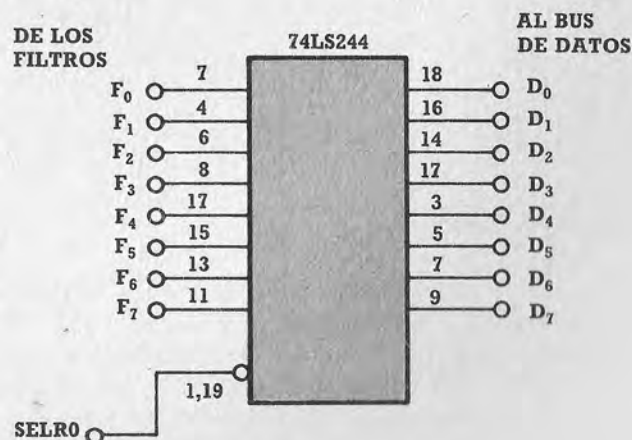


Fig. 9. Circuito de interfaz.

La conexión al ordenador puede hacerse como en los otros dispositivos que hemos diseñado, a través de la tarjeta de ampliación de puertos de entrada salida, utilizando un puerto de entrada para las señales y un bit adicional para el pulso de sincronización. La lectura del número de pasos por cero de la señal requerirá además otro puerto de entrada y una línea de puesta a cero del contador. Las señales de acceso al bus serán seleccionadas, como en circuitos anteriores, con los circuitos de decodificación correspondientes a -SELR0 y SELR1.

Circuito completo

Se muestra el circuito completo con todos los bloques funcionales explicados en los apartados anteriores. Se muestran solamente el primero y el último de los circuitos de filtro. Los demás se definen exactamente igual, con los valores de los componentes según la tabla 1. Los amplificadores operacionales pueden ser de cualquier tipo, pues las frecuencias de trabajo son realmente muy bajas. Es recomendable usar del tipo LM741 en sus versiones de 2 ó 4 amplificadores en un circuito, con objeto de reducir espacio y problemas de cableado.

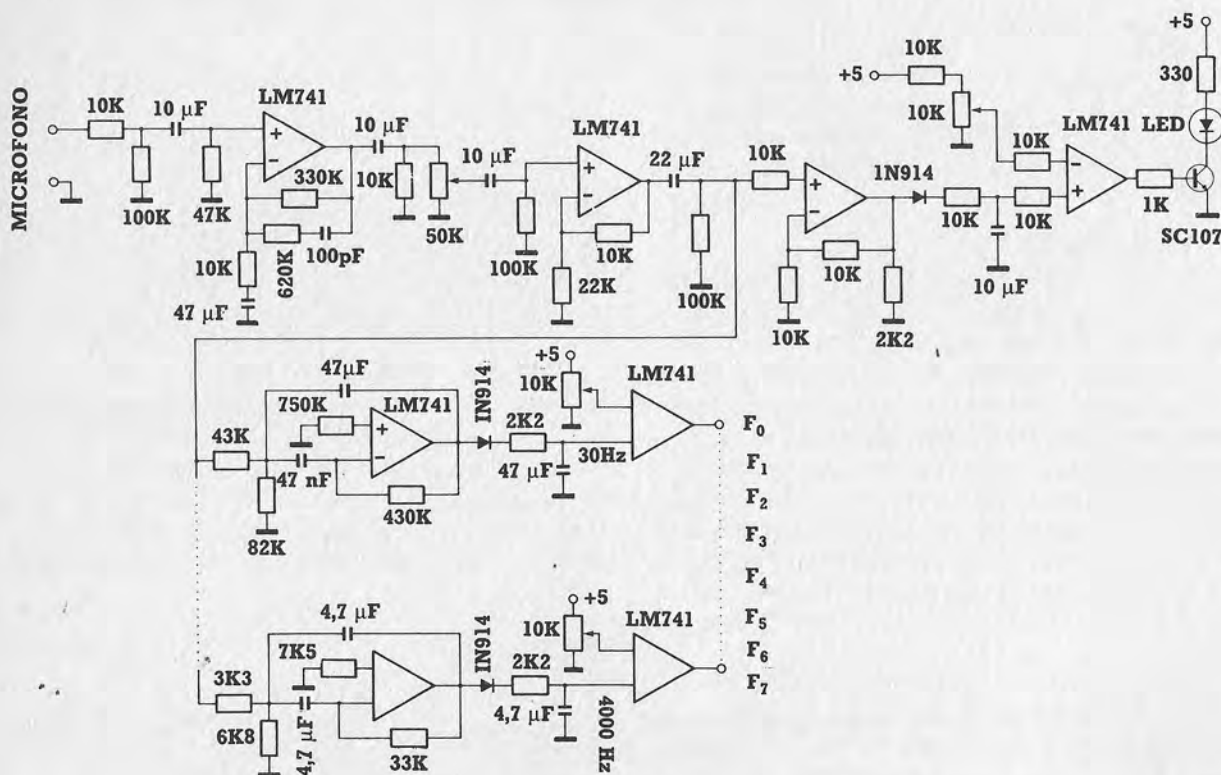


Fig. 10. Circuito completo.

Montaje

La complejidad del circuito es pequeña, pero son muchos los componentes que es necesario utilizar para que el sistema montado tenga aplicación práctica. Para empezar podríamos montar el amplificador de entrada y por lo menos dos etapas de filtro, los correspondientes a las frecuencias en las bandas de

120 y 1000 Hz, que es donde normalmente se concentra la energía de la voz. Con dos filtros y la conexión al ordenador podemos observar la evolución de las componentes en las bandas indicadas y ejercitar los programas que se acompañan.

Cuando ya tengamos práctica con el empleo del sistema, podemos continuar con el montaje de los demás filtros y la medida del número de pasos por cero.

■ Programación de la tarjeta

El programa de adquisición de las señales es muy simple, pues se reduce a muestrear periódicamente y almacenar el resulta-

do que esté disponible en el puerto de entrada. Se lista a continuación un pequeño programa, que puede servir de ejemplo para elaborar uno más completo adaptado a nuestras necesidades.

```
10 REM * * * * *
20 REM PROGRAMA DE ADQUISICION DE SEÑALES
30 REM * * * * *
40 REM UTILIZA LA TARJETA DE FILTROS DE VOZ
50 REM EN EL IBM PC
60 REM * * * * *
70 DEFINT A-Z
80 DIM B(320): REM VECTOR DE MUESTRAS
90 REM DESCRIPCION DEL PUERTO DE ENTRADA:
91 REM BITS 0-7= SALIDAS DE LOS FILTROS
92 REM DESCRIPCION DE PUERTO DE CONTROL:
93 REM BIT 0 = BORRADO DE CONTADOR
94 REM DESCRIPCION DE PUERTO DE ESTADO:
95 REM BIT 0 = TOMAR MUESTRA
96 REM BITS 1-7 = CUENTA DE PASOS POR CERO
100 REM DEFINICION DE PUERTOS
110 PDATA=&H201: REM DATOS DE LOS FILTROS
120 PCONTROL=&H203: REM BYTE DE CONTROL
130 PCUENTA=&203: REM BYTE DE ESTADO Y CUENTA
140 REM MASCARAS PARA PRESENTACION
150 M(0)=1: M(1)=2: M(2)=4: M(3)=8
160 M(4)=16: M(5)=32: M(6)=64: M(7)=128
170 F$(0)=" ": F$(1)=" "
200 KEY OFF
210 CLS
220 PRINT " PROGRAMA DE CAPTACION DE SEÑALES"
230 PRINT
240 PRINT " VERIFIQUE LAS CONEXIONES DE ENTRADA"
250 PRINT
260 PRINT " SELECCIONE OPCION:"
270 PRINT
280 PRINT " 1.-CAPTAR SEÑAL"
290 PRINT
300 PRINT " 2.-PRESENTAR RESULTADO NUMERICAMENTE"
310 PRINT
320 PRINT " 3.-PRESENTACION GRAFICA"
330 PRINT
340 PRINT " 4.-TERMINAR"
350 PRINT
360 INPUT "OPCION=":OPCION
370 IF OPCION<1 OR OPCION>4 THEN BEEP: GOTO 210
380 ON OPCION GOSUB 1000,2000,3000,4000
390 IF OPCION=3 THEN END
400 GOTO 210
990 REM * * * * CAPTAR SEÑALES * * * *
1000 CLS
1100 INPUT "PULSE ENTER CUANDO ESTE PREPARADO":X#
1140 PRINT
1150 PRINT "CAPTANDO SEÑALES"
1160 A=INP(PDATA)
1170 IF A=0 THEN 1160
1200 FOR I=1 TO 320
1210 B(I)=INP(PDATA)
1220 NEXT I
1230 RETURN
1990 REM * * * * PRESENTAR RESULTADO * * * *
2000 CLS
2010 PRINT "LISTADO DE LA SEÑAL CAPTADA"
2020 PRINT
2030 FOR I=1 TO 320
2040 PRINT " ":HEX$(B(I)):
2050 NEXT I
2060 RETURN
2990 REM * * * * PRESENTAR GRAFICAMENTE * * * *
2995 REM * * * * SOBRE FORMATO ALFANUMERICO * * * *
3000 CLS
3010 PRINT "GRAFICA DE LA SEÑAL CAPTADA"
3020 PRINT
3030 FOR K=0 TO 3
3040 FOR J=0 TO 7
3050 FOR I=1 TO 80
3060 PRINT F$(B(I+80*K) AND M(J)):
3070 NEXT I
```

```
3080 NEXT J
3090 PRINT
3100 NEXT K
3200 RETURN
4000 REM TERMINA
4010 OUT PCONTROL:O
4100 RETURN
```

La parte del programa dedicada a la visualización es también sencilla, necesitando simplemente la presentación de cada bit de forma gráfica en unas coordenadas X-Y, en la que la X representa el tiempo y la Y el rango de frecuencias del filtro correspondiente. Si la señal superó el umbral, aparecerá iluminada la casilla; si, por el contrario, estaba en 0, aparecerá el color de fondo.

El programa podría además contemplar el almacenamiento en un fichero de la secuencia de muestras correspondientes a una palabra dada, la presentación gráfica de una secuencia almacenada y la comparación de una secuencia almacenada con una nueva secuencia. Aquí es donde realmente pueden presentarse problemas de programación. Es casi seguro que aunque se pronuncie exactamente igual una palabra dos veces seguidas no se obtendrán las mismas muestras de las señales. Es necesario, por tanto, realizar una comparación con todas las secuencias almacenadas y asignar a la más parecida la correspondiente palabra. El problema es aún mayor para la misma palabra pronunciada pasado un cierto tiempo, aunque sea la misma persona y el mismo lugar. Una comparación visual podrá ayudarnos en un principio, pero para poder hacer algo práctico deberemos utilizar algún algoritmo que determine la "distancia" lógica entre las secuencias de dos palabras.

Para que solamente se almacene información válida por cada palabra, el programa de adquisición deberá explorar hasta que haya alguna muestra con alguna frecuencia por encima del umbral y luego tomará un número fijo de muestras. Antes de almacenar la palabra se eliminarán las muestras del final que contengan solamente 0. El tiempo entre muestras puede controlarse mediante un bucle de programa o mediante la observación de la señal externa. En la versión con intérprete de BASIC el tiempo entre muestras es bastante grande, para que no sea necesario aumen-

tarlo por programa. Para una aplicación real sería conveniente utilizar un programa compilado o en código de máquina.

■ Conclusión

Se ha mostrado un circuito simple que permite presentar de manera gráfica el espectro de una señal de baja frecuencia y su evolución en el tiempo. Utilizado con entrada de voz puede servir para dar comandos al ordenador si incluimos dentro de la aplicación la comparación de las secuencias de muestras con las órdenes de control del programa.

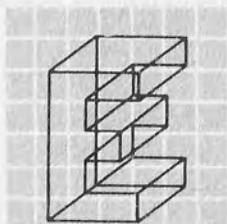
El circuito, aunque no es muy sofisticado, permite iniciarse en el área muy interesante de proceso digital de señales mediante ordenador personal y que trataremos con mayor profundidad en futuros tomos. El empleo de nuestro ordenador personal hace más cómodo, eficiente y utilizable en diversidad de aplicaciones, el experimentar con proceso de señales.

Con el circuito indicado estamos todavía lejos de poder realizar el reconocimiento de voz en términos prácticos, pero lo que sí podemos es adquirir experiencia sobre el problema de análisis espectral de señales de baja frecuencia y una cierta familiarización con el tratamiento de señales mediante ordenador.

En un próximo número veremos las posibilidades de análisis de sistemas mediante su respuesta en frecuencia y posteriormente el empleo de circuitos integrados que permiten realmente hacer el reconocimiento de la voz, todo ello a través de tarjetas que podemos añadir a nuestro ordenador personal a través de puertos de entrada salida, pues el proceso que necesita hacerse en tiempo real se realiza en la tarjeta y todo lo demás en el ordenador.

NATURALEZA Y TECNOLOGÍA

■ Biología: el corazón



El corazón es un órgano musculoso hueco, situado en la cavidad torácica entre los dos pulmones y cuyo peso en el hombre adulto oscila alrededor de los 350 g.

Interiormente consta de dos partes totalmente comunicadas entre sí. Cada una de ellas se divide a su vez en dos cámaras: aurícula y ventrículo.

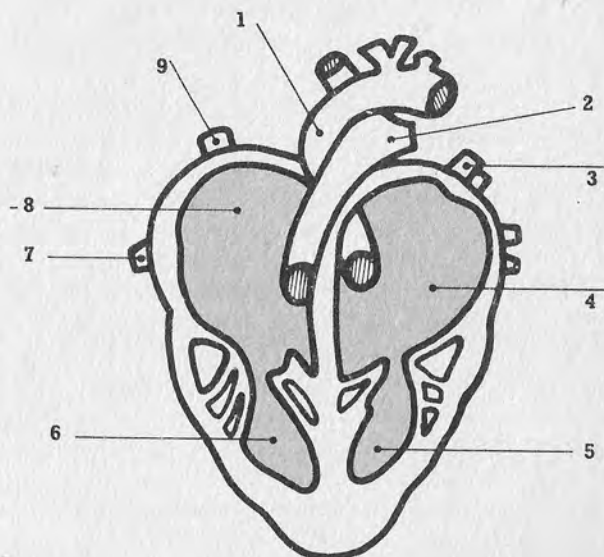
La comunicación de cada aurícula con su correspondiente ventrículo se realiza a través de válvulas membranosas que permiten el paso de la sangre de la aurícula al ventrículo, pero no a la inversa.

El corazón está formado por el miocardio, masa contráctil y muscular, y recubierto por dos membranas; una externa, pericardio, y una interna, miocardio.

Las fibras musculares que constituyen el miocardio tienen la propiedad de contraerse rítmicamente, originando así el latido cardíaco. Los dos movimientos rítmicos que realiza el corazón se denominan sístole (contracción) y diástole (dilatación).

Exteriormente la contracción cardíaca se manifiesta mediante los latidos y ruidos cardíacos, que pueden percibirse con el estetoscopio.

Las arterias, pulmonar y aorta, son los vasos que conducen la sangre desde el corazón a todos los órganos. Las venas, por el contrario, son vasos que ingresan sangre en el corazón, pudiendo ser de dos tipos: venas pulmonares y venas cavas. Estas últimas son la vena cava inferior y la vena cava superior.



- | | |
|--------------------------|------------------------|
| 1. Arteria aorta. | 2. Arteria pulmonar. |
| 3. Venas pulmonares. | 4. Aurícula izquierda. |
| 5. Ventrículo izquierdo. | 6. Ventrículo derecho. |
| 7. Vena cava inferior. | 8. Aurícula derecha. |
| 9. Vena cava superior. | |

Fig. 1.

■ Sobre el programa

En la primera parte del programa se representa un dibujo como el de la figura 1, señalando los aspectos más importantes de la morfología del corazón. Te servirá para retener de un modo más fácil los conceptos anteriormente expuestos.

A continuación deberás responder a las preguntas que te irá formulando el ordenador sobre los puntos esenciales de la estructura y funcionamiento del corazón.

No olvides contestar siempre en minúsculas.

APRENDER CON EL ORDENADOR

```

10 REM *****
20 REM * biologia *
30 REM * el corazon *
40 REM *****
50 CLS
60 ndib=8
70 REM *****
80 REM * puntos del dibujo *
90 REM *****
100 REM
110 REM miocardio
120 REM
130 DATA 39, 107,330,73,342,50,337,35,317,30,290,38,251,56,212,82,186,103,180,12
140 DATA 150,196,172,242,192,295,187,314,171,329,148,334,123,327,104,302
150 DATA 103,280,91,280,91,295,103,324,71,335,59,328,49,317,48,277,48,277
160 DATA 73,159,72,254,53,262,56,242,71,218,97,205,101,206,103,233,84,251,89,252
170 DATA 103,240,104,276
180 REM
190 REM auricula derecha
200 REM
210 DATA 17,121,210,143,215,157,236,145,244,150,250,159,247,170,273,178,297,177,
220 DATA 162,324,137,324,126,318,115,301,116,250,139,256,117,240,120,213
230 REM
240 REM vena cava
250 REM
260 DATA 29, 98,337,97,349,109,363,112,363,104,373,119,380,125,370,130,373,130,3
270 DATA 146,386,146,374,148,374,153,380,156,380,156,373,158,373,166,379
280 DATA 170,376,164,370,172,365,163,354,137,357,140,348,157,348,154,341
290 DATA 134,339,123,328,101,296,95,296
300 REM
310 REM arteria aorta
320 REM
330 DATA 4, 31,290,25,287,18,296,29,303
340 REM
350 REM vena pulmonar
360 REM
370 DATA 4,54,337, 51,346,63,353,70,342
380 REM
390 REM cava inferior
400 REM
410 DATA 4, 97,342,104 ,329, 122,353,134,355
420 REM
430 REM ventriculo derecho
440 REM
450 DATA 4,169,330,184,339,190,324,182,319
460 REM
470 REM auricula izquierda
480 REM
490 DATA 7,191,306,208,306,210,303,196,301,195,294,203,293,203,291
500 REM
510 REM *****
520 REM * representacion grafica *
530 REM *****
540 REM
550 FOR i=1 TO ndib
560 READ nptos
570 READ x,y:PLOT x,y
580 FOR j=1 TO nptos-1
590 READ x,y
600 DRAW x,y
610 NEXT j
620 NEXT i
630 REM *****
640 REM * colocar explicacion *
650 REM *****
660 REM
670 LOCATE 5,17:PRINT "Corazon"
680 LOCATE 5,18:PRINT " "
690 PRINT
700 FOR i=1 TO 6
710 READ e$
720 LOCATE 1,i+19: PRINT STR$(i);".";e$
730 NEXT i
740 DATA Auricula derecha
750 DATA Ventriculo derecho
760 DATA Auricula izquierda
770 DATA Ventriculo izquierdo
780 DATA Arteria pulmonar
790 DATA Arteria aorta
800 REM
810 REM *****
820 REM * rectas *
830 REM *****
840 REM
850 PLOT 135,220:DRAW 230,220
860 PLOT 145,270:DRAW 240,270
870 PLOT 25,220:DRAW 80,220
880 PLOT 20,320:DRAW 75,290
890 REM
900 REM *****
910 REM * situar numeros *
920 REM *****
930 REM
940 LOCATE 16,9:PRINT 3
950 LOCATE 15,12:PRINT 4
960 LOCATE 1,12:PRINT 2
970 LOCATE 1,4:PRINT 1
980 LOCATE 12,1:PRINT 6
990 LOCATE 7,1:PRINT 5
1000 FOR gh=1 TO 3500:NEXT gh
1010 CLS
1020 PRINT
1030 PRINT
1040 PRINT
1050 PRINT
1060 PRINT
1070 REM *****
1080 REM * datos test *
1090 REM *****

```

```

1100 REM
1110 DATA "La arteria","aorta nace en"
1120 DATA "el ventriculo derecho"
1130 DATA "la auricula izquierda"
1140 DATA "el ventriculo izquierdo",3
1150 DATA "Los movimientos ","del corazon son"
1160 DATA "sistole y diastole"
1170 DATA "auricula y ventriculo"
1180 DATA "simpatico y parasimpatico",1
1190 DATA "Las dos","auriculas"
1200 DATA "comunican entre si"
1210 DATA "no se comunican"
1220 DATA "no existen",2
1230 DATA "La membrana externa","del corazon es"
1240 DATA "pericardio","miocardio","endocardio",1
1250 DATA "Los vasos que","ingresan la sangre son"
1260 DATA "las venas cavas","la cava superior","la arteria aorta",1
1270 DATA "Los ruidos cardiacos","se perciben con"
1280 DATA "el fonendoscopio","el microscopio"
1290 DATA "el estetoscopio",3
1300 REM
1310 REM *****
1320 REM * realizar preguntas *
1330 REM *****
1340 REM
1350 FOR f=1 TO 6
1360 READ p$(f,1),p$(f,2)
1370 PRINT p$(f,1)+p$(f,2)
1380 PRINT:PRINT
1390 FOR c=1 TO 3
1400 READ r$(c)
1410 PRINT " ";c;" ";r$(c)
1420 NEXT c
1430 READ res
1440 PRINT
1450 PRINT
1460 PRINT
1470 INPUT "Respuesta";w
1480 PRINT
1490 PRINT
1500 PRINT
1510 REM *****
1520 REM * comprobar respuesta *
1530 REM *****
1540 REM
1550 IF res=w THEN PRINT"Correcto":GOTO 1610
1560 REM *****
1570 REM * temporizador *
1580 REM *****
1590 REM
1600 PRINT"Error, la correcta es";res
1610 FOR ww=1 TO 3500:NEXT ww
1620 CLS
1630 NEXT f

```

Modificaciones para otros equipos

SPECTRUM:

```

570 read x1,y1: let x1=x1*0.425
575 let y1=y1*0.44:plot x1,y1
590 read x2,y2: let x2=x2*0.425
595 let y2=y2*0.44 :draw x2-x1,y2-y1
600 let x1=x2: let y1=y2
670 print at 17,5: "Corazon"
680 print at 18,5: "_____"
720 print at i+19,1:str$(i);". ";e$
850 plot 57,97:draw 40,0
860 plot 62,119:draw 40,0
870 plot 10,97:draw 23,0
880 plot 8,140:draw 23,-13
940 print at 9,16;3
950 print at 12,15;4
960 print at 12,1;2
970 print at 4,1;1
980 print at 1,12;6
990 print at 1,7;5

```

COMMODORE:

```

50 print chr$(147)
570 read x1,y1: x1=x1*0.425: y1=y1*0.48
590 read x2,y2 : x2=x2*0.425: y1=y1*0.48
600 line (x1,y1)-(x2,y2):x1=x2: y1=y2

```

```

850 line(57,105)-(98,105)
860 line (62,129)-(102,125)
870 line (10,105)-(34,105)
880 line (8,154)-(32,140)
1010 print chr$(147)
1620 print chr$(147)

```

Para la simulación de las sentencias locate consulta el tomo 1 de esta colección.

IBM:

```

55 screen 2
570 read x1,y1
590 read x2,y2: line (x1,y1)-(x2,y2)
600 x1=x2:y1=y2
670 locate 17,5
680 locate 18,5
720 locate i+19,1
850 line (135,220)-(230,220)
860 line (145,270)-(240,270)
870 line (25,220)-(80,220)
880 line (20,320)-(75,290)
940 locate 9,16
950 locate 12,15
960 locate 12,1
970 locate 4,1
980 locate 1,12
990 locate 1,7

```

APRENDER CON EL ORDENADOR

En todas las sentencias LOCATE invierte el orden de los argumentos como se indica:

MSX:

```
55 screen 1
570 read x1,y1: x1=x1*0.425 :y1=y1*0.48
590 read x2,y2 : x2=x2*0.425 :y2=y2*0.48
600 line (x1,y1)-(x2,y2):x1=x2 :y1=y2
850 line(57,105)-(98,105)
860 line (62,129)-(102,129)
870 line (10,105)-(34,105)
880 line (8,154)-(32,140)
```

MATEMATICAS

■ Resolución de triángulos rectángulos

Sea ABC un triángulo rectángulo y A su ángulo recto. Llamaremos elementos del triángulo ABC a los lados a, b, c y a sus ángulos opuestos A, B y C.

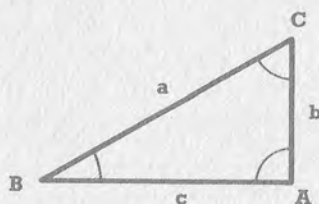


Fig. 2. Elementos de un triángulo rectángulo.

Diremos que los ángulos B y C son dependientes, porque satisfacen la siguiente igualdad: $B + C = 90$

También los lados diremos que son dependientes, pues por el teorema de Pitágoras: $a^2 = b^2 + c^2$

Es posible obtener otras relaciones entre los elementos de un triángulo rectángulo a partir de las definiciones de las razones trigonométricas.

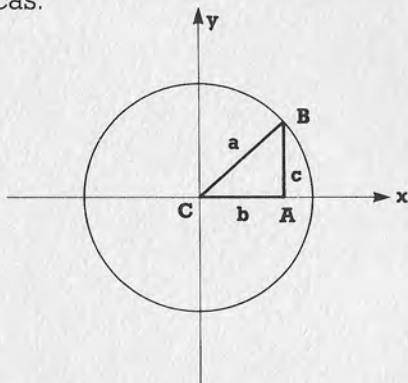


Fig. 3.

De la figura 3 se deducen, por ejemplo, las expresiones:

$$\begin{aligned}\operatorname{sen} C &= c / a \rightarrow c = a \operatorname{sen} C \\ \cos C &= b / a \rightarrow b = a \cos C\end{aligned}$$

Resolver un triángulo rectángulo significa calcular todos sus elementos conociendo solamente dos de ellos.

De los seis elementos de un triángulo uno de ellos, A (el ángulo recto) siempre es conocido y los demás se pueden calcular empleando las relaciones de dependencia.

Según cuales sean los datos, se pueden distinguir los cinco casos siguientes:

1. Conocido un ángulo y la hipotenusa:

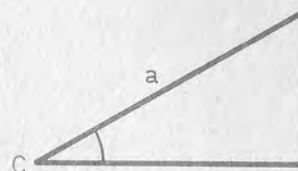


Fig. 4.

2. Conocido un ángulo y un cateto:

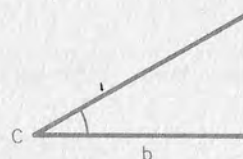


Fig. 5.

3. Conocido un ángulo y el cateto opuesto:

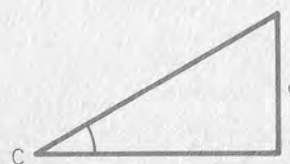


Fig. 6.

4. Conocida la hipotenusa y un cateto:

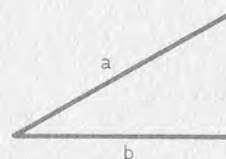


Fig. 7.

5. Conocidos los dos catetos:

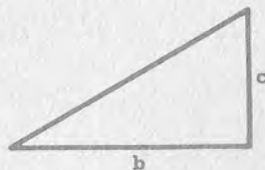


Fig. 8.

datos y te presentará en la pantalla tu triángulo y los valores de todos sus elementos.

Introduce el valor de los ángulos en grados. Si prefieres operar en radianes, sólo tienes que modificar las líneas 370, 520 y 670, en las que se realiza la conversión de grados a radianes.

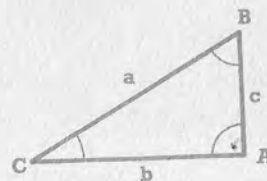
Un ejemplo de resolución en la pantalla sería el de la figura 9.

El programa

El programa te ofrece la posibilidad de resolver rápidamente cualquier triángulo rectángulo. Solamente tienes que introducir los

Datos

a = 32
b = 16



Incógnitas

C = 60°
B = 30°
C = 27,7

Fig. 9.

```

10 REM *****
20 REM * resolucio de
30 REM * triangulos rectangulos *
40 REM *****
50 CLS
60 PRINT
70 PRINT
80 PRINT
90 REM *****
100 REM * casos posibles *
110 REM *****
120 REM
130 PRINT "Resolucion de triangulos rectangulos"
140 PRINT "conocidos solo dos datos"
150 PRINT
160 PRINT TAB(5);"1.un angulo y la hipotenusa"
170 PRINT TAB(5);"2.un angulo y un cateto"
180 PRINT TAB(5);"3.Un angulo y el cateto opuesto"
190 PRINT TAB(5);"4.La hipotenusa y un cateto"
200 PRINT TAB(5);"5.Los dos catetos"
210 PRINT
220 INPUT "Selecciona una opcion";s
230 IF s>5 OR s<1 THEN GOTO 220
240 CLS:GOSUB 1100
250 PRINT
260 PRINT
270 PRINT
280 PRINT "Datos";TAB(8);"Incognitas"
290 PRINT
300 REM *****
310 REM * primer caso *
320 REM *****
330 REM
340 IF s>1 THEN GOTO 490
350 INPUT "C=";c1
360 INPUT "a=";a
370 LET c2=c1*PI/180
380 LET b1=90-c1
390 LET b=a*COS(c2)
400 LET c=a*SIN(c2)
410 LOCATE 8,6:PRINT "B=";b1
420 LOCATE 8,7:PRINT "b=";b
430 LOCATE 8,8:PRINT "c=";c
440 GOTO 1030
450 REM *****
460 REM * segundo caso *
470 REM *****
480 REM
490 IF s>2 THEN GOTO 640
500 INPUT "C=";c1
510 INPUT "b=";b
520 LET c2=c1*PI/180
530 LET b1=90-c1
540 LET a=b/COS(c2)
550 LET c=b*TAN(c2)
560 LOCATE 8,6:PRINT "B=";b1
570 LOCATE 8,7:PRINT "a=";a
580 LOCATE 8,8:PRINT "c=";c
590 GOTO 1030
600 REM *****
610 REM * tercer caso *
620 REM *****
630 REM
640 IF s>3 THEN GOTO 790
650 INPUT "C=";c1
660 INPUT "c=";c
670 LET c2=c1*PI/180
680 LET b1=90-c1
690 LET a=c/SIN(c2)
700 LET b=c/TAN(c2)
710 LOCATE 8,6:PRINT "B=";b1
720 LOCATE 8,7:PRINT "a=";a
730 LOCATE 8,8:PRINT "b=";b

```

```

740 GOTO 1030
750 REM *****
760 REM * cuarto caso *
770 REM *****
780 REM
790 IF s>4 THEN GOTO 940
800 INPUT "a=";a
810 INPUT "b=";b
820 IF a<b THEN PRINT "Datos erroneos":GOTO 800
830 LET c=SQR(a*a-b*b)
840 LET c1=ATN(c/b)*180/PI
850 LET b1=90-c1
860 LOCATE 8,6:PRINT "B=";b1
870 LOCATE 8,7:PRINT "C=";c1
880 LOCATE 8,8:PRINT "c=";c
890 GOTO 1030
900 REM *****
910 REM * quinto caso *
920 REM *****
930 REM
940 INPUT "b=";b
950 INPUT "c=";c
960 LET c1=ATN(c/b)*180/PI
970 LET nn=b*b+c*c
980 LET a=SQR(nn)
990 LET b1=90-c1
1000 LOCATE 8,6:PRINT "C=";c1
1010 LOCATE 8,7:PRINT "B=";b1
1020 LOCATE 8,8:PRINT "a=";a
1030 PRINT:PRINT:PRINT
1040 PRINT:PRINT
1050 PRINT:PRINT
1060 INPUT "Deseas realizar otra operacion";n$
1070 IF r$(">"$1) THEN END
1080 CLS:GOTO 60
1090 REM *****
1100 REM * subrutina de dibujo *
1110 REM *****
1120 REM
1130 ORIGIN 370,220
1140 DRAW 80,0:DRAW 80,120:DRAW 0,0
1150 LOCATE 24,8:PRINT "a"
1160 LOCATE 30,8:PRINT "b"
1170 LOCATE 26,13:PRINT "c"
1180 LOCATE 1,1
1190 RETURN
    
```

Modificaciones

COMMODORE:

50 print chr\$(147)
 240 print chr\$(147)
 1130 no se pone
 1135 line (158,105)-(192,105)
 1140 line (192,105)-(192,153)
 1145 line (192,153)-(158,105)
 1080 print chr\$(147)

La simulación de las sentencias locate la encontrarás en el tomo 1 de esta colección.

IBM:

35 screen 2
 líneas 410, 560, 710, 860, 1000 locate 6,8
 líneas 420, 570, 720, 870, 1010 locate 7,8
 líneas 430,580,730,880,1020 locate 8,8
 1130 no se pone
 1135 line (370,220)-(450,220)
 1140 line (450,220)-(450,320)
 1145 line (450,320)-(370,220)
 1150 locate 8,24
 1160 locate 8,30
 1170 locate 13,26

MSX:

35 screen 1
 1130 no se pone

1135 line (158,105)-(192,105)
 1140 line (192,105)-(192,153)
 1145 line (192,153)-(158,105)

SPECTRUM:

líneas 410, 560, 710, 860 print at 6,8; "B=";b1
 420 print at 7,8;"b=";b
 430, 580, 880 print at 8,8;"c=";c
 570, 720 print at 7,8;"a=";a
 730 print at 8,8;"b=";b
 870 print at 7,8;"C=";c1
 1000 print at 6,8;"C=";c1
 1010 print at 7,8;"B=";b1
 1020 print at 8,8;"a=";a
 1070 goto 9999
 1130 no se pone
 1140 plot 35,97: draw 34,0: draw 0,53:draw -34, -53
 1150 print at 8,24;"a"
 1160 print at 8,30;"b"
 1170 print at 13,26;"c"
 1180 print at 1,1,""

SOCIEDAD

Las declinaciones latinas

Una diferencia fundamental del latín respecto del español es la existencia de decli-

nación en los sustantivos, adjetivos y pronombres.

La función gramatical que desempeñan las palabras en la oración sólo es reconocible en español por su colocación dentro de la frase, por el sentido o por el empleo de las preposiciones.

En latín, el orden de las palabras en la frase no tiene en sí valor gramatical; éste va indicado por la terminación de la palabra.

A las diferentes formas que presenta una misma palabra en sus terminaciones las llamamos casos. En latín son seis: nominativo, vocativo, acusativo, genitivo, dativo y ablativo.

El nominativo desempeña el oficio de sujeto y el de predicado nominal.

El acusativo es el caso del complemento directo de la oración transitiva. A diferencia del español, no lleva nunca preposición.

Un sustantivo puede ir complementado por otro en caso genitivo. Este debe traducirse al español anteponiéndole la preposición de. Por eso, al genitivo se le llama complemento del nombre o complemento nominal.

El dativo realiza la función de complemento indirecto del verbo, indica la persona a quien se atribuye la acción verbal.

El ablativo expresa las diversas circunstancias que acompañan a la acción verbal, pudiendo ir acompañado de preposición.

Declinar es expresar los diferentes casos de una misma palabra. El número de declinaciones es cinco. Es decir, todas las palabras latinas declinables siguen uno de estos cinco tipos de declinaciones.

En la primera declinación agrupamos las palabras (sustantivos y adjetivos) que terminan en el nominativo de singular en "a" y en el genitivo del mismo número en "ae".

Ejemplos: rosa (rosa).
regina (reina)

La segunda declinación comprende los nombres que acaban el nominativo de singular en "us", "er" y "um" y el genitivo en "i".

Ejemplos: puer (niño);
telum (dardo)

La tercera declinación agrupa los sustantivos y adjetivos que terminan el genitivo

de singular en "is"; el nominativo no presenta terminación única.

Ejemplos: amor (amor);
labor (trabajo)

Los sustantivos que terminan el genitivo de singular en "us" pertenecen a la cuarta declinación. El nominativo acaba en "us" o "u".

Ejemplos: genu (rodilla);
portus (puerto)

Los sustantivos incluidos en la quinta declinación terminan el nominativo en "es" y el genitivo en "ei".

Ejemplos: res (objeto, asunto);
dies (día)

■ Sobre el programa

El programa permite la declinación de una palabra perteneciente a cualquiera de las cinco declinaciones.

Debes indicar a qué declinación pertenece esa palabra, seleccionando una de las opciones que se te ofrecen por su número.

Cuando introduzcas el nominativo de la palabra que deseas declinar, hazlo siempre en minúsculas.

Dada la diversidad de nombres pertenecientes a la tercera declinación, con el fin de no alargar excesivamente el programa sólo se consideran como pertenecientes a la tercera declinación aquéllos cuyo tema termina en consonante líquida (*l* o *r*).

Un ejemplo del funcionamiento del programa, con el sustantivo latino rosa (la rosa), sería el siguiente:

Primera declinación

	Singular	Plural
nominativo	rosa	rosae
vocativo	rosa	rosae
acusativo	rosam	rosas
genitivo	rosae	rosarum
dativo	rosae	rosis
ablativo	rosa	rosis

```

10 REM *****
20 REM * declinaciones latinas *
30 REM *****
40 CLS
50 REM
60 REM casos
70 REM
80 FOR i=1 TO 6
90 READ a$(i)
100 NEXT i
110 REM *****
120 REM * terminaciones de los casos *
130 REM *****
140 REM
150 DATA "nominativo","vocativo","acusativo"
160 DATA "genitivo","dativo","ablativo"
170 REM
180 REM primera
190 REM
200 DATA "a","ae","a","ae","am","as","ae","arum","ae","is","a","is"
210 REM
220 REM segunda
230 REM
240 DATA "us","i","e","i","um","os","i","orum","o","is","o","is"
250 DATA "i","i","um","os","i","orum","o","is","o","is"
260 DATA "um","a","um","a","um","a","i","orum","o","is","o","is"
270 REM
280 REM tercera
290 REM
300 DATA "","es","","es","em","es","is","um","i","ibus","e","ibus"
310 REM
320 REM cuarta
330 REM
340 DATA "us","us","us","us","um","us"
350 DATA "us","uum","ui","ibus","u","ibus"
360 DATA "u","ua","u","ua","u","ua","us","uum","ui","ibus","u","ibus"
370 REM
380 REM quinta
390 REM
400 DATA "es","es","es","es","em","es","ei"
410 DATA "erum","ei","ebus","e","ebus","ei"
420 REM
430 REM *****
440 REM * programa principal *
450 REM *****
460 REM
470 PRINT "      Declinaciones"
480 PRINT "-----"
490 PRINT
500 PRINT "      1.Primeras"
510 PRINT
520 PRINT "      2.Segunda"
530 PRINT
540 PRINT "      3.Tercera"
550 PRINT
560 PRINT "      4.Cuarta"
570 PRINT
580 PRINT "      5.Quinta"
590 PRINT
600 INPUT "Selecciona una opcion";n
610 IF n<1 OR n>5 THEN GOTO 600
620 CLS
630 INPUT "palabra latina";l$
640 LET l=LEN(l$)
650 CLS
660 IF n>1 THEN GOTO 930
670 REM
680 REM *****
690 REM * primera declinacion *
700 REM *****
710 REM
720 PRINT "primera declinacion"
730 PRINT "-----"
740 PRINT
750 PRINT TAB(13);"singular";TAB(24);"plural"
760 PRINT TAB(13);"-----";TAB(24);"-----"
770 PRINT
780 LET y$=LEFT$(l$,l-1)
790 FOR f=1 TO 6
800 PRINT a$(f);
810 READ p$
820 PRINT TAB(13);y$+p$;
830 READ p$
840 PRINT TAB(24);y$+p$
850 PRINT
860 NEXT f
870 GOTO 1860
880 REM
890 REM *****
900 REM * segunda declinacion *
910 REM *****
920 REM
930 IF n>2 THEN GOTO 1180
940 LET b$=MID$(l$,l-1,2)
950 IF b$="us" THEN s=12:LET y$=LEFT$(l$,l-2)
960 IF b$="er" THEN LET s=24:LET y$=l$
970 IF b$="um" THEN LET s=36:LET y$=LEFT$(l$,l-2)
980 PRINT "segunda declinacion"
990 PRINT "-----"
1000 PRINT
1010 PRINT TAB(13);"singular";TAB(24);"plural"
1020 PRINT TAB(13);"-----";TAB(24);"-----"
1030 FOR k=1 TO s:READ s$:NEXT k
1040 FOR k=1 TO 6
1050 PRINT a$(k);
1060 READ s$
1070 PRINT TAB(13);y$+s$;
1080 READ s$

```

```

1090 PRINT TAB(24);y$+s$
1100 PRINT
1110 NEXT k
1120 GOTO 1860
1130 REM
1140 REM *****
1150 REM * tercera declinacion *
1160 REM *****
1170 REM
1180 IF n>3 THEN GOTO 1400
1190 PRINT "tercera declinacion"
1200 PRINT "-----"
1210 PRINT
1220 PRINT TAB(13);"singular";TAB(24);"plural"
1230 PRINT TAB(13);"-----";TAB(24);"-----"
1240 LET y$=l$
1250 FOR k=1 TO 48:READ t$:NEXT k
1260 FOR i=1 TO 6
1270 PRINT a$(i);
1280 READ t$
1290 PRINT TAB(13);y$+t$;
1300 READ t$
1310 PRINT TAB(24);y$+t$
1320 PRINT
1330 NEXT i
1340 GOTO 1860
1350 REM
1360 REM *****
1370 REM * cuarta declinacion *
1380 REM *****
1390 REM
1400 IF n>4 THEN GOTO 1670
1410 PRINT "cuarta declinacion"
1420 PRINT "-----"
1430 PRINT
1440 PRINT TAB(13);"singular";
1450 PRINT TAB(24);"plural"
1460 PRINT TAB(13);"-----";
1470 PRINT TAB(24);"-----"
1480 LET b$=RIGHT$(l$,2)
1490 IF b$="us" THEN LET s=60:LET y$=LEFT$(l$,1-2):GOTO 1520
1500 LET y$=LEFT$(l$,1-1)
1510 LET s=72
1520 FOR k=1 TO s:READ c$:NEXT k
1530 FOR i=1 TO 6
1540 PRINT a$(i);
1550 READ c$
1560 PRINT TAB(13);y$+c$;
1570 READ c$
1580 PRINT TAB(24);y$+c$
1590 PRINT
1600 NEXT i
1610 GOTO 1860
1620 REM
1630 REM *****
1640 REM * quinta declinacion *
1650 REM *****
1660 REM
1670 PRINT "quinta declinacion"
1680 PRINT "-----"
1690 PRINT
1700 PRINT TAB(13);"singular";
1710 PRINT TAB(24);"plural"
1720 PRINT TAB(13);"-----";
1730 PRINT TAB(24);"-----"
1740 LET y$=LEFT$(l$,1-2)
1750 FOR k=1 TO 84:READ q$:NEXT k
1760 FOR i=1 TO 6
1770 PRINT a$(i);
1780 READ q$
1790 PRINT TAB(13);y$+q$;
1800 READ q$
1810 PRINT TAB(24);y$+q$
1820 PRINT
1830 NEXT i
1840 PRINT:PRINT
1850 REM
1860 REM *****
1870 REM * repetir proceso *
1880 REM *****
1890 REM
1900 PRINT "Quieres declinar otro sustantivo"
1910 RESTORE
1920 INPUT r$
1930 IF r$="s" THEN GOTO 20
1940 GOTO 1940

```

Modificaciones para otros equipos

El programa es válido para Amstrad, IBM y MSX. Si tu ordenador es un Spectrum o un Commodore, debes hacer los siguientes cambios:

SPECTRUM:

```

5 dim a$(6,10)
780 let y$=l$(1 to 1-1)
940 let b$=l$(1-1 to 1)

```

```

950 let y$=l$(1 to 1-2)
970 let y$=l$(1 to 1-2)
1480 let b$=l$(1-1 to 1)
1490 let y$=l$(1 to 1-2)
1500 let y$=l$(1 to 1-1)
1740 let y$=l$(1 to 1-2)

```

COMMODORE:

```

40 print chr$(147)
620 print chr$(147)
650 print chr$(147)

```

APRENDER CON EL ORDENADOR

PARA LOS MAS JOVENES

■ Adivinar dos números

Aquí tienes la oportunidad de demostrar que eres capaz de adivinar los dos números que el ordenador ha pensado para ti.

El ordenador te dará tres pistas para que intentes descubrir los números: su suma, su diferencia y los valores entre los que se encuentran.

Con estos datos debes ser capaz de descubrir los números que ha pensado tu ordenador. Indica tu respuesta separando am-

bos números por una coma tal como se indica en el siguiente ejemplo:

Ejemplo:

Estoy pensando dos números
entre 1 y 10,

La suma de los números es 15.

Su diferencia es 3.

¿Cuáles son los números?

Respuesta: 9,6.

Si te equivocas en la respuesta el ordenador te lo indicará y te enseñará los números que ha pensado.

Del mismo modo, si contestas correctamente, te felicitará y te propondrá jugar otra vez.

```
10 REM *****
20 REM * adivinar los numeros *
30 REM * pensados por el ordenador *
40 REM *****
50 CLS
60 PRINT "estoy pensando dos numeros"
70 PRINT "entre 1 y 10 "
80 PRINT
90 PRINT
100 REM *****
110 REM * obtencion de los numeros *
120 REM *****
130 REM
140 LET n1=INT(RND*10)+1
150 LET n2=INT(RND*10)+1
160 IF n2>n1 THEN GOTO 80
170 REM
180 REM *****
190 REM * escritura de mensajes *
200 REM *****
210 PRINT "la suma de estos numeros es";n1+n2
220 PRINT "la diferencia es";n1-n2
230 PRINT
240 PRINT
250 INPUT "cuales son esos numeros";c1,c2
260 PRINT
270 REM *****
280 REM * comprobacion de resultados *
290 REM *****
300 REM
310 IF (c1=n1 OR c1=n2) AND (c2=n1 OR c2=n2) THEN PRINT "Muy bien":GOTO 460
320 PRINT "te has equivocado"
330 PRINT
340 PRINT
350 PRINT
360 REM *****
370 REM * solucion *
380 REM *****
390 PRINT "la solucion es n1=";n1;" y n2=";n2
400 PRINT
410 PRINT
420 REM *****
430 REM * retardador *
440 REM *****
450 REM
460 FOR g= 1 TO 2000
470 NEXT g
480 GOTO 50
```

Modificaciones para otros equipos

El programa es válido para Amstrad, IBM y Spectrum. Si tu ordenador es un Commodore o MSX, deberás hacer las siguientes modificaciones:

COMMODORE:

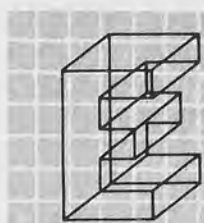
```
50 print chr$(147)
140 let n1=int(rnd(1)*10)+1
150 let n2=int(rnd(1)*10)+1
```

MSX:

```
140 let n1=int(rnd(1)*10)+1
150 let n2=int(rnd(1)*10)+1
```

PEQUEÑA HISTORIA DE LA INFORMATICA

LA MAQUINA ANALITICA (PRIMERA MAQUINA UNIVERSAL)



N 1832, Babbage había llevado a cabo muchos proyectos de todo tipo. La máquina de diferencias le había absorbido gran parte de su tiempo, pero no se sentía satisfecho. La idea primitiva, de la que había partido para diseñarla era pobre, más aún, ramplona, y, por tanto, el futuro de la máquina sería efímero, sin interés.

A pesar de toda las subvenciones de organismos relacionados con el Gobierno, Babbage empezó a olvidarse poco a poco de su «obra de juventud» interesándose por un proyecto verdaderamente revolucionario. La máquina analítica. Esta máquina es mucho más ambiciosa que la anterior, es más, no tiene casi nada que ver con ella. La máquina analítica fue pensada como un computador universal, es decir, lo que hoy se llama «de uso general». (Es importante que aclaremos bien lo que significa «de uso general». Lo haremos, dando ejemplos del término contrario, «uso especial». En este epígrafe podemos incluir máquinas que realizan operaciones específicas, como, por ejemplo, los ordenadores que hoy en día controlan básculas, otros incorporados a coches, para controlar ciertas características especiales, etc.).

La máquina de Babbage estaba pensada para resolver cualquier necesidad de cómputo, y además era totalmente automática.

Veamos cómo concibió Babbage que debía ser su máquina analítica:

En primer lugar, debía disponer, claro está, de una unidad aritmética, para realizar este tipo de operaciones.

Además, debía disponer de una Unidad de Control, que se ocupara de que las operaciones se efectuaran en las secuencias correctas. Esta unidad supone la primera innovación

sobre las máquinas tradicionales, en las que esta labor era llevada a cabo por los operadores.

Un sistema para almacenar los números, tanto aquéllos que se van generando durante las operaciones, como los que constituyen el punto de partida del problema, es decir, los datos suministrados a la máquina. Esta innovación fue absolutamente revolucionaria.

Una unidad de entrada de los datos (considerando como datos los números y las instrucciones que se dan a la máquina para que los procese).

Una unidad de salida, que muestre los resultados de la forma adecuada.

La memoria, naturalmente, era mecánica, y estaba formada por engranajes con ruedas que podía permanecer en una de las diez posiciones posibles, almacenando de este modo un dígito. Esta memoria podía tener una capacidad de unos 1.000 datos, cantidad enorme para la época. Los datos podían estar representados cada uno por un máximo de cincuenta posiciones decimales.

Babbage jamás llamó memoria a su sistema de almacenamiento. Pensaba en él más bien como un granero, o despensa, que como un dispositivo en cierto modo semejante a la «memoria humana».

El sistema de transmisión de los datos a la memoria, también a través de engranajes y varillas, era verdaderamente complicado. Babbage pensó en que la máquina pudiera tomar los datos de la memoria, y también en que fuera posible introducirlos a mano (aunque esta posibilidad retrasaría enormemente el tiempo de ejecución, a pesar de la lentitud de los engranajes mecánicos).

El sistema de control de la máquina era muy ingenioso. Utilizaba tarjetas perforadas y varillas. Cuando las varillas no encontraban una perforación en la ficha, no se movía, ele-

PEQUEÑA HISTORIA DE LA INFORMATICA

vándose las demás. Este sistema fue copiado del que utilizaba Jacquard para controlar los diseños de tejidos en su telar, y del que hablaremos con mucho más detalle. Digamos, sin embargo, que el sistema era bastante sencillo, y permitía una versatilidad enorme. Las fichas perforadas (naturalmente sin sus varillas asociadas) se han utilizado hasta hace muy poco, en los Centros de Proceso de Datos.

Las perforaciones se llevan a cabo según un determinado código, y van dispuestas secuencialmente sobre una cinta, pasando a continuación sobre un tambor.

Babbage pensó en utilizar dos tipos de fichas, de las que antes hemos hablado, y a las que llamó «Fichas de operación» y «Fichas de variables». El primer grupo de fichas (las de operación) tenían como objetivo controlar el funcionamiento de la Unidad Aritmética, especificando además el tipo de operación que debía llevarse a cabo. El segundo grupo de fichas controlaban la transferencia de los datos desde la memoria a la unidad de cálculo, y viceversa. (Hasta hace pocos años, se utilizaba un sistema de tarjetas perforadas muy semejante, uniendo las tarjetas de datos y de instrucciones en un mismo paquete, de tarjetas «de programas».

Las tarjetas utilizadas en la máquina de Babbage eran muy semejantes a las que se han utilizado en Informática, pero algo mayores, y con unas perforaciones sensiblemente mayores también, aunque pequeñas (algo más de un centímetro) si consideramos los sistemas mecánicos que las manipulaban.

El sistema de introducción de los datos también permitía introducir tablas; como sabemos en la época (e incluso en nuestros días) muy útiles para multitud de estudios y trabajos.

Las tablas tenía una aplicación inmediata debido a que evitaban realizar innumerables cálculos, laboriosos. Sin embargo, Babbage supo intuir que en el futuro dejaran de utilizarse, o se utilizaran menos, ya que las operaciones de cálculo serían tan rápidas que es probable que fuera más sencillo calcular una cantidad (mediante su fórmula) que obtenerla de una tabla, comprobando su exactitud. (Respecto de este último punto, la máquina de Babbage disponía de unos timbres que sonaban cada vez que la máquina necesitaba algún dato, o cuando el dato introducido estaba equivocado.)

El sistema de presentación de los datos permitía un tipo de impresión directa, median-

te moldes de los que se vaciaban bloques para la impresora. Este sistema también utilizaba tarjetas perforadas.

Simplificando un poco, digamos que el sistema de salida grababa los datos en una plancha de cobre, para poder imprimirlos con posteridad.

La máquina de Babbage hubiera sido una revolución en su tiempo. Pero nunca llegó a terminarse. El proyecto era demasiado ambicioso, y requería resolver innumerables pequeños problemas técnicos, de resistencia de materiales, de rigidez, etc. Babbage trabajó en su máquina el resto de su vida, dedicándole todo su interés. También invirtió en ella todo su dinero, y el obtenido del Gobierno. Pero nunca llegó a ver su obra, lo que le amargó, y dio a sus últimos años una componente de odio hacia la humanidad que no lo había comprendido ni ayudado. La realidad es que le era difícil explicar cómo se había gastado su vida y su fortuna sin poder mostrar el resultado de sus trabajos. Mas aún, todos ellos estaban inacabados, produciendo al visitante, o al amigo una sensación de pesadumbre e incompreensión.

Los diseños de Babbage eran revolucionarios, y los medios para llevarlos a cabo, complicados, debido a las técnicas utilizadas en la época. Sin embargo, Babbage no fue un visionario. Era un hombre práctico, el tiempo se ha encargado de darle la razón. Sus inventos y diseños fueron prácticos y útiles. Otras de las innovaciones estudiadas y logradas por Babbage que aplicó a su máquina fue un «sistema de arrastres anticipados». En este caso, el término «arrastre» se refiere a los dígitos obtenidos de las operaciones, que se van añadiendo en la siguiente posición más significativa. Estos arrastres se van produciendo «en cascada», creando problemas en el sistema de engranajes. Babbage consiguió que todos los arrastres fueran simultáneos, mediante un sistema llamado de «arrastre anticipado». Este sistema se basa en que siempre que el total de la suma de dos números excede de nueve, se produce un arrastre. Esto ocurre del mismo modo en decenas, decenas de millar, etc. Aunque el orden consecutivo se mantiene en la lógica del proceso, en la práctica se puede conseguir que estos arrastres se produzcan casi simultáneamente. Babbage utilizó este mismo razonamiento para dotar a sus máquinas de la capacidad de tomar decisiones lógicas. Así, pues, la máquina de Babbage no solamente efectuaba todo tipo de operaciones.

¿Sabía usted que...

¿Sabía usted que uno de los lenguajes con más futuro dentro de la Informática es, a su vez, uno de los más antiguos?

En efecto, el lenguaje de programación LISP es uno de los que más se está desarrollando y más se extiende en la actualidad por haber sido concebido específicamente para el procesamiento de listas (LISP = List Processing) y ser, por tanto, muy apto para la programación declarativa propia de las nuevas áreas de Inteligencia Artificial. De hecho, hace un lustro era tal el desarrollo de diferentes dialectos de LISP (casi uno para cada máquina y cada empresa) que se tuvo que imponer un estándar (Common Lisp) para poder comunicar resultados de una instalación a otra. Sin embargo, LISP es uno de los lenguajes más antiguos (de entre los llamados lenguajes de alto nivel): las primeras versiones datan de 1960 (hace veintisiete años: ¡La prehistoria de los lenguajes de programación!) y una versión depurada casi como las actualmente utilizadas (en LISP puro, no dialectos) se publicó en 1962 por parte de John McCarthy (creador del LISP en los laboratorios del MIT, Massachusetts Institute of Technology).

También podía llevar a cabo ciertas decisiones lógicas. Fue la primera máquina «seria» que podía tomar este tipo de decisiones, ya que, aunque existían autómatas capaces de hacerlo, nunca se había considerado esta posibilidad para resolver problemas de cálculo. Así, cuando la máquina recibe una instrucción lógica, debe elegir el camino a seguir, dependiendo del paso previo. Este tipo de operaciones condicionales suponen una enorme innovación. El sistema que pensó Babbage para resolver el problema se basa también en el «arrastre anticipado», llamado también principio de «complemento de nueves». Así, si tenemos dos números, uno mayor que otro, y restamos el mayor del menor, considerando un número fijo de dígitos, tendremos lo siguiente:

$$\begin{array}{r} 0000156 \\ - 0000237 \\ \hline 9999819 \end{array}$$

El conjunto de nueves obtenidos es resultado de la «toma» de unos. Al aumentar el número de dígitos, el conjunto nueves se iría extendiendo a la izquierda. En una sustracción normal, el resultado sería -81. La cifra anterior (9999819) sería el «complemento de nueves» de -81. Es fácil ver que esta operación de sus-

tracción y es equivalente a añadir 9999819 al número descartar el 1 del arrastre final. (El lector puede comprobarlo.) Babbage lo utilizó para que los arrastres fueran «simultáneos» y también en la toma de decisiones, ya que al obtenerse respuestas negativas, se producirían conjuntos de nueves a la izquierda hasta



llenar las posiciones e incluso más. Babbage dispuso otra posición especial, que activaría determinados engranajes que a su vez actuarían una parte de la calculadora..., que a su vez tomaría determinado conjunto de tarjetas perforadas. La posibilidad de que la máquina cambiara la secuencia de operaciones como consecuencia de alguno de los resultados previos, era una característica nueva y poderosísima. Colocando un registro especial, donde se indicara el número de veces que se debía realizar una determinada operación, la máquina iría sustrayendo unidades de ese registro cada vez que realizara la operación. El punto de corte sería cuando la máquina provocara un número negativo en el registro, y el «conjunto de nueves complementarios» se utilizarían para detener la operación y pasar a otra, etcétera.

Babbage escribió bastante sobre sus proyectos. Sin embargo, no se explicaba con demasiada claridad, y sus escritos no tienen

PEQUEÑA HISTORIA DE LA INFORMATICA

demasiado interés. Muchos otros hombres inquietos de la época (de varias nacionalidades) se interesaron por sus trabajos, y publicaron escritos y ensayos, a veces mucho más interesantes que los del propio Babbage. Entre este grupo de personas, debemos mencionar a Lady Lovelace, considerada la primera programadora, y amiga personal de Babbage, y a otro ingeniero joven italiano, Menabrea, que dio un ciclo de conferencias publicando informes sobre la obra de Babbage.

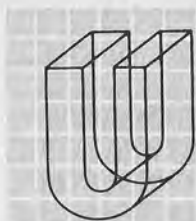
La máquina de Babbage es precursora de los ordenadores actuales, una especie de «vieja abuelita» que camina despacio, pero que llega a la meta. Babbage había calculado que su máquina podría desarrollar una velocidad de unas sesenta sumas por minuto (velocidad increíble). Como la máquina tenía tantos engranajes, también consideró la posibilidad de que la energía necesaria para el movimiento de engranajes, etc., se efectuara mediante vapor.

Entre las personas que más ayudaron a Babbage se encuentra Lady Lovelace. De esta dama hablaremos en otro momento, pero no debemos olvidar que fue una magnífica matemática, y que se interesó notablemente en los proyectos de Babbage, ayudándole en todos los campos, difundiendo sus trabajos, programando su máquina e incluso ayudándole a pensar posibles fuentes de ingresos, cuando la fortuna del inventor apenas si le permitía continuar comprando material para su máquina.

Babbage había diseñado una máquina superior a sus posibilidades. Murió en 1871, habiendo invertido toda su fortuna en la empresa, y sin apenas ver resultados, ya que la máquina no se construyó totalmente, y sólo podían admirarse algunos componentes en la máquina del inventor. (Hoy pueden verse en Londres.)

Su hijo Henry Babbage, presentó la máquina en 1910 a la «Astronomical Society».

LOS LENGUAJES DEL FUTURO: PROLOG



N aspecto capital del desarrollo de la informática ha sido la creación y potenciación de los lenguajes de programación. El elemento básico que ha marcado la evolución de la informática ha resultado ser, desde luego,

el desarrollo del hardware en sus dos aspectos de obtención de componentes cada vez más sofisticados y de mejora de la arquitectura y de los ordenadores. Pero también ha contribuido a dicha evolución (y ha venido potenciada a su vez, por ella) la mejora de los lenguajes de programación. Por esto, pensar en la Informática del futuro es pensar (junto a la arquitectura de las máquinas) en los lenguajes de programación.

De entre los lenguajes de futuro sobresale el PROLOG en dos sentidos: por un lado, es un lenguaje muy apropiado para los sistemas de inferencia que están en la base de la mayoría de las aplicaciones de Inteligencia Artificial; de hecho, los dos lenguajes de programación más utilizados en este área son LISP y PROLOG, y los entornos más sofisticados que se están desarrollando en los últimos años se basan en alguno (o ambos) de estos dos lenguajes. Por otro lado, el PROLOG ha sido tomado como base del (de los) lenguaje(s) de programación a desarrollar en el proyecto japonés «del ordenador de quinta generación».

¿Quiere esto decir que PROLOG va a ser el lenguaje de programación del año 2000? Esto es difícil de contestar, pero lo que parece claro es que PROLOG o algún dialecto del será un lenguaje básico en los ordenadores que en esa fecha estemos utilizando.

Y ¿Por qué PROLOG precisamente y no otro lenguaje de programación? Ya hemos comentado que no es sólo PROLOG el lenguaje

a utilizar en Inteligencia Artificial, sino uno de los posibles. Todos ellos tienen una característica especial: son lenguajes declarativos y no procedurales. Esto, como veremos con detalle más adelante, supone varias ventajas prácticas, además de la facilidad general que aportar para la representación del conocimiento y el proceso de la información en tareas de inferencia: estas ventajas son que cada porción de un programa (cada proposición o conjunto de ellas) es evaluable independientemente del resto y, por otro lado, que las modificaciones son fáciles de realizar, precisamente por esta independencia de cada proposición.

El concepto de lenguaje declarativo (en contraposición a los lenguajes procedurales) responde a la idea de facilitar al ordenador la información de que disponemos (algo estructurada, claro está) y dejar que él la maneje para obtener los resultados que deseamos, en vez de tener que concebir nosotros el procedimiento de cálculo (algoritmo) y explicárselo al ordenador, como se hace en los lenguajes convencionales.

En concreto, en PROLOG lo que realizamos en «programación lógica» (PROLOG = PROgraming LOGic): un programa el PROLOG es un conjunto de proposiciones lógicas que el intérprete es capaz de manejar. Por ejemplo, podemos decirle que Carlos I fue el padre de Felipe II y que Felipe el Hermoso fue padre de Carlos I; si además, le decimos que el «padre del padre de alguien» es el abuelo de esa persona, podemos inmediatamente pasar a preguntarle si Carlos I es padre de alguien, si Felipe el Hermoso es abuelo de alguien, quién es el padre de Felipe II, etc., y el propio sistema hará sus «cálculos» y nos contestará. (Por el contrario, en un lenguaje más convencional —el popular BASIC, por ejemplo— habría que definir las variables, teniendo

TEMAS MONOGRAFICOS DE VANGUARDIA

do cuidado con su tipo y definir cómo podemos evaluar si alguien es padre o no de alguien y ver cómo comparamos las cadenas de caracteres y otro sinfín de detalles que en el PROLOG obviamos.

Además, otra ventaja del PROLOG es que su sintaxis es muy cómoda y sencilla, y que sus expresiones son casi como en castellano. Más concretamente, podemos construir una «base de datos» en PROLOG, introduciendo las siguientes proposiciones:

CarlosI es-padre-de Felipe II
FelipeH es-padre-de Carlos I

y la siguiente regla de inferencia:

x es-abuelo-de z si x es-padre-de y
& y es-padre-de z

y pasar a preguntar:

es (CarlosI es-abuelo-de FelipeII)?
es (FelipeH es-abuelo-de FelipeII)?
quién (x : x es-abuelo-de FelipeII)?

(la última expresión se leería: «quién es x tal que x es el abuelo de Felipe II»; es decir, ¿quién es el abuelo de Felipe II?).

Las respuestas a estas tres preguntas son evidentes para una persona cualquiera, pero ¿cómo trabaja PROLOG? Para aclararlo, vamos a introducir alguna proposición más a nuestra base de datos (con Margarita de Parma, hija natural de Carlos I y su hijo Alejandro Farnesio, que sucedió a Juan de Austria como gobernador de los Países Bajos, con Isabel Clara Eugenia hija predilecta de Felipe II, regente (pero no gobernadora) de los Países Bajos y con el propio Juan de Austria, también gobernador de los Países Bajos):

CarlosI es-padre-de FelipeII
CarlosI es-padre-de MargaritaP
CarlosI es-padre-de JuanA
FelipeII es-padre-de IsabelCE
MargaritaP es-madre-de AlejandroF
AlejandroF fue-gobernador
JuanA fue-gobernador
IsabelCE fue-regente

(obsérvese que ha aparecido una nueva relación, el predicado «es-madre-de» y dos predicados adicionales: «fue-gobernador» y «fue-regente»).

Además, debemos completar la definición del abuelo con el nuevo predicado aparecido («es-madre-de») quedando:

x es-abuelo-de z si x es-padre-de y
& y es-padre-de z
x es-abuelo de z si x es-padre de y
& y es-madre-de z

Si ahora queremos saber quién fue una persona cuyo abuelo fue Carlos I y fue gobernador de los Países Bajos, preguntaremos:

quién (x : CarlosI es-abuelo-de x & x fue-gobernador)

PROLOG procesará nuestra pregunta de izquierda a derecha e intentará averiguar quién cumple la primera condición indicada: Carlos I es-abuelo-de x. Para ello accede a las reglas de inferencia de que dispone y ve que esa condición es equivalente a que se cumpla una cualesquiera de los dos siguientes (nuevos subjetivos a obtener):

CarlosI es-padre-de y & y es-padre-de z (1)
o bien CarlosI es-padre-de y & y es-madre-de z

Toma la primera de ellas y ve, en la base de datos, si hay algún elemento «y» que cumple esta primera condición; hay tres: Felipe II, Juan de Austria y Margarita de Parma; averigua si el primero de ellos (el primer valor «y» correspondiente a un hijo de Carlos I) cumple la segunda condición: en efecto, «Felipe II es-padre-de IsabelCE». Por tanto, ya tenemos un elemento x que cumple CarlosI es-abuelo-de x. Pero, desgraciadamente, Isabel-

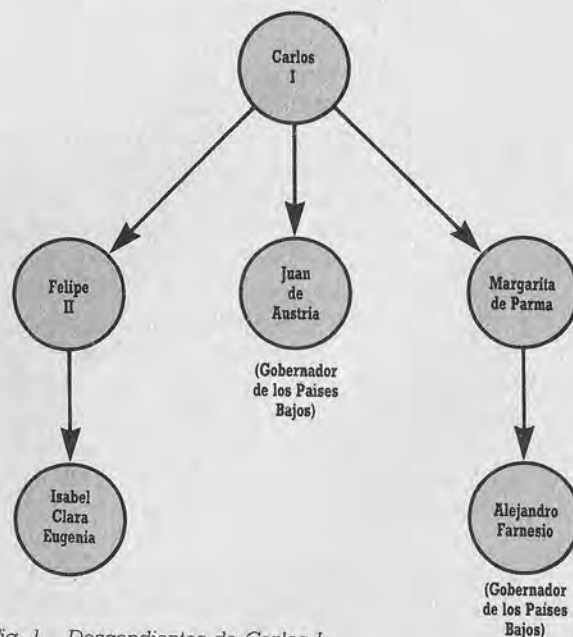


Fig. 1. Descendientes de Carlos I.

CE no cumple la segunda condición que imponemos en la pregunta, pues no fue gobernadora de los Países Bajos. Entonces PROLOG tiene que volver atrás (backtracking se llama al proceso) para seguir con alguno de los casos posibles, que ha desechado anteriormente.

Toma entonces la segunda opción (JuanA) que cumplía «CarlosI es-padre-de y», pero este elemento no cumple «y es-padre-de z», pues en la base de hechos no aparece JuanA como padre de nadie.

Así, vuelve (backtrack) a Margarita de Parma que había cumplido la primera condición impuesta: CarlosI es-padre-de y. Para

este nuevo valor de «y», preguntamos por la segunda parte de las condiciones impuestas como subobjetivos en (1) y vemos que MargaritaP no cumple tampoco, pues «MargaritaP es-padre-de z» no es cierto para ningún z. Esta rama del proceso se ha agotado sin éxito y hay que volver atrás (backtracking) preguntando por la segunda opción (subobjetivo) de las previstas en (1). Ahora PROLOG verá que «CarlosI es-padre-de MargaritaP» y «MargaritaP es-madre-de AlejandroF» (habiendo comprobado y desechado «CarlosI es-padre-de FelipeII» y «CarlosI es-padre-de JuanA», pues ninguno de los dos cumple «y es-madre-de z»).

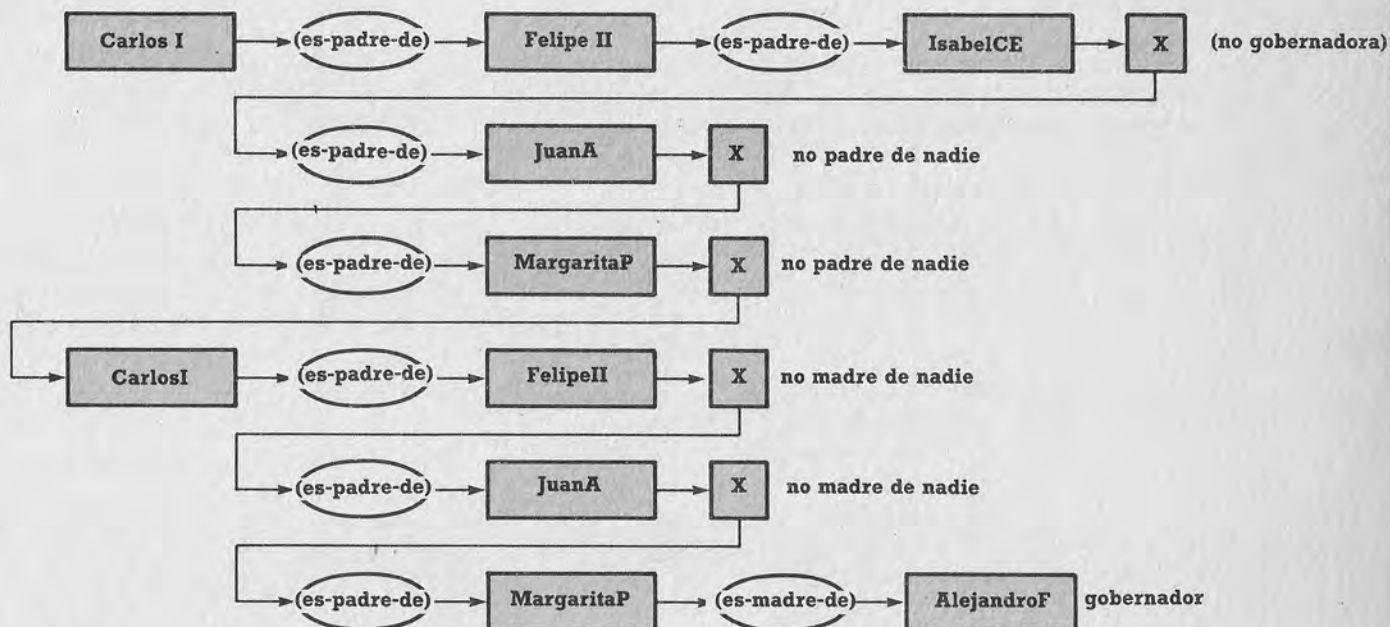


Fig. 2. Proceso de Backtracking.

Con este nuevo valor obtenido comprueba la segunda condición de la pregunta inicial y ve que, en efecto, Alejandro Farnesio fue gobernador de los Países Bajos: entonces PROLOG contesta:

x = AlejandroF

y como, después de las oportunas comprobaciones y vueltas atrás, descubre que no hay ningún otro elemento que responda a la pregunta, añade:

no hay (más) respuestas

Estos conceptos son los que el propio

Colmerauer*, coinventor del PROLOG, sintetiza en lo que llama el «reloj del PROLOG» (o máquina del PROLOG) y que aparece en la figura 3.

En ella, «i» contiene un número no negativo que representa el tiempo.

«Ci» contiene las restricciones (o condiciones) que deben ser satisfechas en el tiempo (momento) «i».

«Ti» incluye la secuencia de términos que no han sido eliminados aún en el momento «i».

* Proceedings of the Eighth International Joint Conference on Artificial Intelligence. Karlsruhe, W. Germany. Tomo 1, pág. 496.

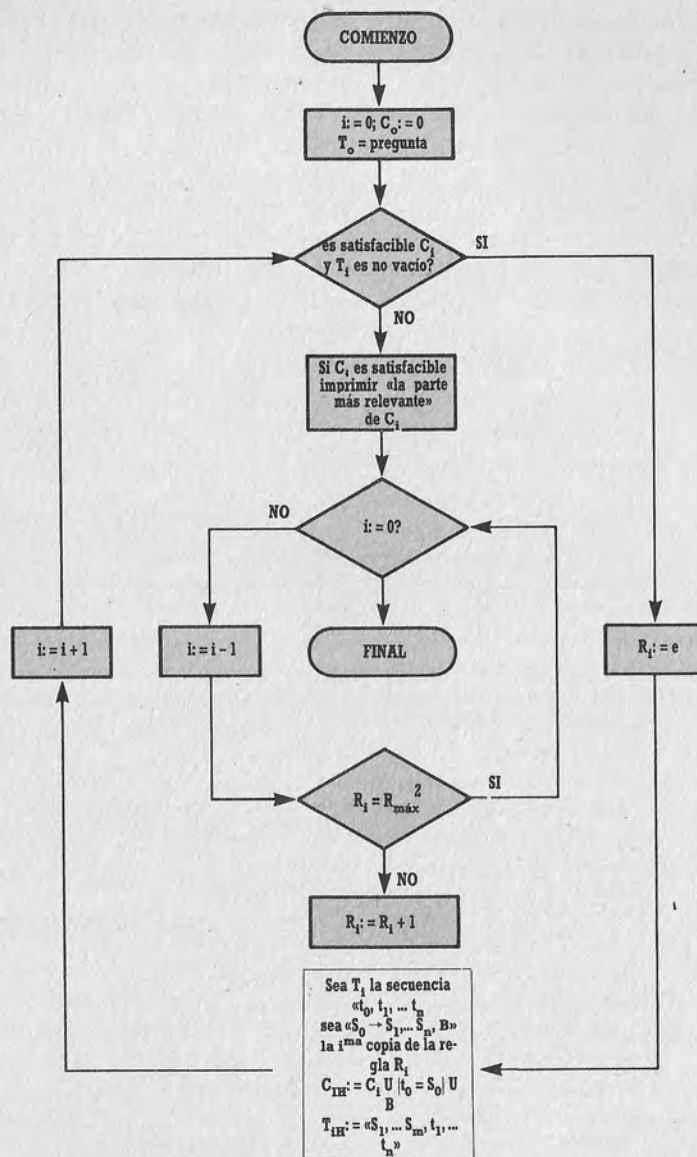


Fig. 3. El reloj del PROLOG.

«R_i» es el número de la regla elegida en el tiempo «i».

«R_{máx}» indica el número de la última regla.

«la parte más relevante de C_i» significa la subcondición (en forma reducida) que se refiere a las variables que aparecen en la pregunta.

El funcionamiento de la máquina viene descrito por los dos círculos de la figura 3. En el exterior, el proceso gira una vuelta en el sentido de las agujas del reloj, mientras el tiempo («i») se incrementa en una unidad; en el interior, se produce un giro en sentido contrario a las agujas del reloj, mientras el tiempo

decrece una unidad. Se puede cambiar la progresión del tiempo pasando de uno a otro a través de uno de los dos «puentes» de un solo sentido que hay uniendo ambos círculos.

La ejecución de un programa PROLOG consiste en responder a una pregunta representada por un término (al principio se almacena en T_o). Todas las respuestas que se van computando son restricciones o condiciones en las cuales el término representa hechos específicos. Comenzamos el proceso por el par (C_o, T_o); C_o está vacío y T_o es la pregunta de comienzo. Cada vuelta que se da al círculo exterior incrementa la condición vigente actualmente «C_i» y transforma la secuencia «T_i».

Si la regla que se utiliza ya contiene una condición Be, esta condición ese añade al juego de condiciones elementales de que disponemos. Concluye el proceso en cuanto se genera una condición no satisfacible o cuando la secuencia «Ti» queda vacía: es el momento en que se realiza el paso de backtracking (vuelta atrás) para tomar otras reglas. Si en algún momento, durante el proceso de giro del «reloj», «Ci» es satisfacible, se imprime la respuesta (y vuelve atrás para localizar respuestas adicionales).

Para concluir esta breve presentación del PROLOG incluimos un programa escrito en el dialecto más popular de los existentes para microordenadores pequeños (el microPROLOG) y referente al «análisis» de una «base de hechos» geográfica*.

a) Se utilizan las relaciones:
 <ciudad> capital-de <provincia>
 <provincia> provincia-de <autonomía>
 <ciudad> situación (<posición-vertical> <posición-horizontal>)

referidas estas posiciones a su situación en un mapa imaginario. Como nombres de individuos se utilizarán las capitales de las provincias, las provincias de tres Autonomías españolas y estas tres Autonomías (Asturias, Rioja y País Vasco).

b) La base de datos podría ser:
 Oviedo capital-de Asturias
 Bilbao capital-de Vizcaya
 SSebastián capital-de Guipúzcoa
 Vitoria capital-de Alava
 Logroño capital-de Rioja
 Vizcaya provincia-de PaísVasco
 Guipúzcoa provincia-de PaísVasco
 Alava provincia-de PaísVasco
 Asturias provincia-de AsturiasAut
 Rioja provincia-de RiojaAut
 Oviedo situación (15 5)
 Bilbao situación (16 12)
 Sebastián situación (17, 15)

Vitoria situación (13, 13)
 Logroño situación (11, 14)

c) Con todo esto, podíamos formular preguntas del tipo:

1. ¿Qué ciudades están al norte de Vitoria?

wich (x : x situación (y z) and Vitoria situación (Y Z) and Y LESS

o bien en la versión castellana de microPROLOG:

que (x : x situación (y z) e Vitoria situación (Y Z)

e Y es-menor-que y

(Nótese que en la versión castellana del microPROLOG de Spectrum la conectiva lógica «and» se ha traducido por «e» en vez de «y», para no confundirla con esta variable).

2. ¿Hay alguna provincia del País Vasco cuya capital esté al norte de Logroño y al sur de Oviedo?

is (x provincia-de PaísVasco and y capital-de x and y situación (z X) and Logroño situación (Y Z) and Oviedo situación (x₁ y₁) and y LESS z and z LESS x₁)

o en castellano:

? (x provincia-de País Vasco e y capital-de x e y situación (z X) e Logroño situación (Y Z) e Oviedo situación (x₁ y₁) e Y es-menor-que z e z es-menor-que x₁)

3. ¿En qué ciudad y Autonomía hay una ciudad al sur y al este de Bilbao?

wich (x y : x provincia-de y and z capital-de x and z situación (X Y) and Bilbao situación (Z x₁) and X LESS Z and x₁ LESS Y)

y en castellano

que(x y : x provincia-de y e z capital-de x e z situación (XY) e Bilbao situación (Z x₁) e Xesmq Z e x₁ esmq Y)

(utilizando la abreviatura «esmq» en vez de la frase completa es-menor-que).

* Inspirado en el manual del microPROLOG, disponible para Spectrum y preparado por K. L. Clark, F. G. McCabe y J. R. Ennals.

■ GLOSARIO DE TERMINOLOGIA UTILIZADA EN PROGRAMACION LOGICA

Backtracking. Proceso de vuelta atrás. En el desarrollo de la búsqueda de soluciones realizada en el árbol lógico de la base de hechos, cuando en una rama no se obtiene una solución válida, es necesario «volver atrás» por el árbol hasta la última condición comprobada, para seguir el proceso de búsqueda por otra rama.

Base de hechos. Conjunto de proposiciones y reglas de inferencia sobre el que se realiza la búsqueda de soluciones.

Condiciones o restricciones. Cada uno de los elementos que conforman la base de hechos. Conjunto de aseveraciones que constituyen el conocimiento que, sobre el tema de que se trata, se dispone.

Conectiva u operador lógico. Cada uno de los elementos de conexión que aparecen entre las proposiciones simples para componer las proposiciones más complejas que aparecen en la base de hechos.

Conocimientos, base de. Concepto equivalente al de base de hechos.

Declarativo, lenguaje. Lenguaje de programación basado en la mera presentación de los hechos (o conocimiento) y no en la

definición de los algoritmos de proceso, como sucede en los lenguajes procedurales (los lenguajes convencionales).

Inferencia, regla de. Regla que permite obtener informaciones o hechos adicionales a partir de los que ya están disponibles. Se habla también de **Proceso de inferencia** o **Sistema de inferencia**.

Predicado. Cualidad que se atribuye a un sujeto o relación existente entre varios sujetos (en el primer caso se llama predicado monádico y en el segundo predicado poliádico). Uno o varios predicados con su(s) sujeto(s) correspondiente(s) constituyen una proposición.

Procedural. Lenguaje de programación concebido para definir algoritmos, en contraposición a los lenguajes declarativos.

Proposición o aserción. Afirmación simple que constituye la base de conocimientos. Cada información que compone dicha base de conocimientos o base de hechos.

Subobjetivo. Cada uno de los objetivos en que se descompone un objetivo básico que haya que evaluar, en virtud de alguna de las reglas de inferencia de la base de conocimientos.

VOCABULARIO DE INFORMATICA

Inclusión. Operador lógico que tiene la siguiente propiedad: Si P y Q son dos aseveraciones, P inclusión Q es falso, si P es verdadero y Q falso, y es verdadero si son verdaderas las dos aseveraciones. Este operador suele indicarse como $P \supset Q$, y tienen el mismo significado que la implicación «Si Entonces».

Inconector. En diagramas de flujo, conector que indica la continuación de una línea de flujo rota.

Incremental, ordenador. Ordenador en el que se utiliza con frecuencia su característica de representar los datos de forma incremental. Ordenador diseñado especialmente para procesar cambios en las variables, y también sus valores absolutos.

Incremental, representación. Método de representación en el que lo que se representa son los cambios de los valores de una variable, no los propios valores.

Incrementar. Acción por lo que se va añadiendo una cantidad a otra.

Index. (Véase **Indice**.)

Indexada, dirección. Dirección que se ve modificada por el contenido de un registro índice antes o durante la ejecución de una instrucción al ordenador.

Indice, registro. Registro cuyo contenido se suma o se sustrae de la dirección del operando antes o durante la instrucción del ordenador. De esta forma se pueden calcular

en las instrucciones en lenguaje máquina las direcciones absolutas de las distintas posiciones de la memoria central, basta con sumar la dirección base (contenida en el registro) más el valor del desplazamiento relativo.

Indice. Referencia ordenada del contenido de un fichero o documento, junto con otras referencias para su identificación y localización de su contenido.

Perteneciente a un registro índice. Símbolo que acompaña a un número y que sirve para identificarlo como formando parte de una matriz, etc., si la matriz está representada por los elementos $x_1, x_2, \dots, x_{10}, 1, 2, \dots, 10$ son los índices.

Indirecta, dirección. Dirección que especifica una posición de almacenamiento en la que se encuentra una dirección directa, u otra indirecta. Tienen el mismo significado de dirección multinivel.

Industrial, proceso de datos. Proceso de datos, específicamente diseñado para actividades industriales.

Infija, notación. Método de creación de expresiones matemáticas que consiste, en alternar los operandos simples y los operadores. Los términos se definen de forma que cumplan las leyes de precedencia y paréntesis de los operadores. El operador, a su vez, realiza su función sobre los términos adyacentes. En notación infija, la expresión «a más b multiplicado por c» se escribiría como $(a + b) \times c$. En contraposición con notación con prefijos (o notación prefija).

VOCABULARIO DE INFORMATICA

Información, bits de. En telecomunicación, bits generados por la fuente de datos, y que el sistema de transmisión de datos no utiliza para el control de errores.

Información, sistema de retroalimentación de la. En telecomunicación, sistema de transmisión de la información que utiliza un «eco» para verificar la pureza de la transmisión.

Información, proceso de la. Término que tiene el mismo significado que proceso de datos.

Información, recuperación de la. Métodos y procedimientos para volver a disponer de información desde los sistemas de almacenamiento.

Información, separador de la. Grupo de caracteres de control que se utilizan como frontera para separar la información. El tipo o nombre del separador no indica en absoluto qué es lo que separa.

Información, teoría de la. Rama del conocimiento que estudia la pureza de las transmisiones (siempre sujetas a fallos de transmisión, distorsión y ruido), las condiciones de propagación, la cantidad de información que se transmite por un canal, los tipos de códigos, etc.

Informática. Contracción de «INFORMación AutoMÁTICA». Indica todo lo relacionado con el proceso de datos, los ordenadores, periféricos, etc., en el sentido más general.

Information retrieval. (Ver **Recuperación de la información.**)

Inhibir. Evitar que se produzca alguna acción como resultado de otra. Por ejemplo, se pueden tomar medidas para inhibir la presentación en pantalla de determinado tipo de información considerada reservada.

Inhibición, señal de. Señal que impide que se lleve a cabo determinada operación.

Inicialización. Procedimiento que establece los valores iniciales necesarios para que el dispositivo comience a funcionar correctamente. Un ejemplo puede ser atribuir valores a las variables de un programa. Los intérpretes del BASIC moderno inicializan automáticamente, poniendo a cero todas las variables, y a valor nulo todas las cadenas.

Inicialización de un disco. Procedimiento por el cual se prepara un disco utilizado con anterioridad, para volver a utilizarlo. Los disquetes y discos duros necesitan ser inicializados grabando en ellos determinadas indicaciones. A esta operación se le denomina generalmente formateado, y a veces, sectorización.

Initial program loader (IPL). Programa que realiza la carga automática de un sistema operativo u otro programa, de forma que proceda bajo su propio control.

Inmediato. Modo de trabajo en BASIC, en el que las instrucciones no están numeradas y se ejecutan inmediatamente, pulsando detrás de cada una la tecla Return, para que el ordenador sepa que se ha terminado de introducir la instrucción. Puede indicar también un modo de direccionamiento utilizado en Ensamblador para referirse a una constante contenida en la propia instrucción. También puede indicar una operación o acceso a la memoria de ejecución directa.

Input. (Véase **Introducción de datos.**)

Input área. Zona de almacenamiento reservada para una entrada. Idéntico a **Input block.**

Input, channel. Canal de un estado en un dispositivo o elemento lógico.

Input data. (Ver **Introducción de los datos.**)

Input, proceso. Proceso por el cual un dispositivo recibe los datos. Proceso por el cual un equipo periférico o sistema de almacenamiento externo envía los datos al dispositivo de almacenamiento interno.

Instrucción. Orden que especifica una determinada operación, con los valores o direcciones de sus operandos. Está escrita en un determinado lenguaje. Las palabras cla-

ve que aparecen en las líneas de programa son llamadas a su vez instrucciones o comandos.

Instrucción, dirección de. Dirección que se utiliza para tomar una instrucción.

Instrucción, código de. Tiene el mismo significado que **Código de operación.**

Instrucción, contador de. Contador que indica la posición de la siguiente instrucción que debe ser ejecutada.

